ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Επίλυση ΜΔΕ με την Μέθοδο των Γραμμών σε Κατανεμημένα Συστήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Νικόλαου-Ηρακλή Μήτση

Βόλος, Οκτώβριος 2008

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Επίλυση ΜΔΕ με την Μέθοδο των Γραμμών σε κατανεμημένα Συστήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΝΙΚΟΛΑΟΥ-ΗΡΑΚΛΗ ΜΗΤΣΗ

Επιβλέπων : Τσομπανοπούλου Παναγιώτα
Επίκουρη Καθηγήτρια Τ.Μ.Η.Υ.Τ.Δ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή τον Οκτωβρίου 2008.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Παναγιώτα Τσομπανοπούλου Επίκουρη Καθηγήτρια Τ.Μ.Η.Υ.Τ.Δ.	Ηλίας Χούστης Καθηγητής Τ.Μ.Η.Υ.Τ.Δ.	Εμμανουήλ Βάβαλης Αναπληρωτής Καθηγητής Τ.Μ.Η.Υ.Τ.Δ.

Βόλος, Οκτώβριος 2008

(Υπογραφή)

.....

ΝΙΚΟΛΑΟΣ-ΗΡΑΚΛΗΣ ΜΗΤΣΗΣ

Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων Πανεπιστημίου Θεσσαλίας

Ευχαριστίες

Ύστερα από μια πορεία πέντε και πλέον ετών στο Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας ολοκληρώνω τις προπτυχιακές μου σπουδές με την εκπόνηση της παρούσας διπλωματικής εργασίας.

Θα ήθελα να ευχαριστήσω θερμά την κα. Τσομπανοπούλου Παναγιώτα Επίκουρη Καθηγήτρια του Τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών & Δικτύων, και τον κ. Εμμανουήλ Βάβαλη, Αναπληρωτή Καθηγητή του Τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών & Δικτύων, τόσο για την βοήθεια και τις κατευθύνσεις που μου προσέφεραν, ως επιβλέποντες της διπλωματικής μου εργασίας, όσο και για την συμπαράστασή τους, ως καθηγητές και φίλοι.

Ευχαριστώ επίσης τον συνεπιβλέποντα της εργασίας μου, κ.Ηλία Χούστη, Καθηγητή του Τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών & Δικτύων, για την καθοδήγησή του.

Ακόμα, θα ήθελα να ευχαριστήσω από καρδιάς τον κ. Δημήτρη Συρίβελη, υποψήφιο Διδάκτορα του Τμήματος Μηχανικών Η/Υ, Τηλεπικοινωνιών & Δικτύων, για τις γνώσεις που μου παρείχε γύρω από το cluster της σχολής.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου για την αμέριστη συμπαράσταση της όλα αυτά τα χρόνια, όπως και τους συμφοιτητές και τους φίλους με τους οποίους πορεύτηκα όλα αυτά τα χρόνια σε όμορφες και πάνω από όλα έντονες στιγμές, που θα συνοδεύουν πάντα τις μνήμες της φοιτητικής μου ζωής.

Περίληψη

Στην παρούσα διπλωματική εργασία ασχολούμαστε με την επίλυση γραμμικών, μερικών διαφορικών εξισώσεων, δευτέρας τάξης (Second Order, Linear Partial Differential Equations) με την μέθοδο των γραμμών (Mol – Method Of Lines) και την ανάπτυξή της παράλληλης εφαρμογής της.

Πιο συγκεκριμένα, χωρίζουμε τη διπλωματική σε δύο τμήματα. Στο πρώτο και θεωρητικό (2^o Κεφάλαιο ανάπτυξης), κάνουμε μια ανάλυση των εννοιών που μας απασχολούν. Συγκεκριμένα, ξεκινάμε την παρουσίαση, από την ευρύτερη έννοια των μερικών διαφορικών εξισώσεων και μελετάμε τα χαρακτηριστικά της ομάδας των εξισώσεων, που μας απασχολεί. Έπειτα, αναλύουμε τη σειριακή μέθοδο των γραμμών (Mol-C), μίας αναλυτικής μεθόδου επίλυσης του συγκεκριμένου συνόλου μερικών διαφορικών εξισώσεων, υλοποιημένη στη γλώσσα προγραμματισμού C.

Περνώντας στο πρακτικό κομμάτι της εργασίας (3^o Κεφάλαιο ανάπτυξης), υλοποιούμε την παράλληλη εφαρμογή της μεθόδου των γραμμών. Την υλοποίηση την πραγματοποιούμε στη γλώσσα προγραμματισμού Python, ενώ εκμεταλλευόμαστε την σειριακή Mol-C εφαρμογή, που αναλύουμε στο θεωρητικό κεφάλαιο ανάπτυξης. Το πρόγραμμα που αναπτύσσουμε, το εφαρμόζουμε στο παράλληλο υπολογιστικό περιβάλλον Centaurus, που αποτελεί και το cluster του τμήματος. Από τις μετρήσεις που παίρνουμε, εξάγουμε συμπεράσματα σχετικά με την βελτίωση της απόδοσης της μεθόδου των γραμμών, ως παράλληλης εφαρμογής, συγκριτικά με την αντίστοιχη σειριακή.

Η διπλωματική αυτή μπορεί να γίνει οδηγός για την πραγματοποίηση αποδοτικών παράλληλων εφαρμογών σε Python, που ο σειριακός τους αλγόριθμος είναι ήδη υλοποιημένος. Αυτό συμβαίνει καθώς τα εργαλεία που χρησιμοποιούμε και οι τεχνικές βρίσκουν απόλυτη εφαρμογή σε τέτοιου είδους προσπάθειες.

Λέξεις Κλειδιά: <<Μερικές Διαφορικές Εξισώσεις, Μέθοδος των Γραμμών, Παράλληλα Υπολογιστικά Συστήματα.>>

Πίνακας περιεχομένων

1

Εισαγωγή

Σε αυτό το κεφάλαιο εισάγουμε τον αναγνώστη στο θέμα και τις έννοιες τις διπλωματικής εργασίας που αναπτύξαμε. Στο εισαγωγικό αυτό κεφάλαιο σκοπεύουμε να δώσουμε στον αναγνώστη μια πλήρη εικόνα των όσων θα αναπτύξουμε στα επόμενα κεφάλαια.

1.1 Αριθμητική Επίλυση ΜΔΕ σε Κατανεμημένα Συστήματα

Ο ευρύς και συνεχώς αναπτυσσόμενος τομέας των παράλληλων υπολογισμών έχει προξενήσει μεγάλο ενδιαφέρον στην επιστημονική κοινότητα. Πολλές υλοποιημένες σειριακές εφαρμογές ακόμα και αλγόριθμοι, που σχεδιάζονται εξ αρχής και αποβλέπουν στην παράλληλη εκτέλεση τους, επωφελούνται από τις υπολογιστικές δυνατότητες που προσφέρουν τα συστήματα παράλληλης επεξεργασίας.

Τα παράλληλα υπολογιστικά συστήματα έχουν τη δυνατότητα να συσπειρώσουν και να εκμεταλλευτούν πολλαπλές υπολογιστικές μονάδες. Αυτές οι μονάδες μπορούν να είναι: 1.ατομικοί υπολογιστές με πολλαπλούς πυρήνες επεξεργασίας (multi – core processors), 2.επεξεργαστές που συντονίζονται και λαμβάνουν εργασίες εκτέλεσης από έναν κεντρικό, ενώ επικοινωνούν μεταξύ τους για ανταλλαγή πληροφοριών, μέσω συστημάτων μεταφοράς δεδομένων (distributed or cluster computers), 3.πολλοί διαφορετικοί υπολογιστές,

συνδεδεμένοι στο διαδίκτυο, που μοιράζονται επιμέρους-ανεξάρτητες εργασίες ενός, μεγάλης κλίμακας, προβλήματος (Grid).

Από τους πόρους, ευρείας κλίμακας, που προσφέρουν τα υπολογιστικά συστήματα επωφελούνται πολλές επιστήμες, όπως αυτή των φυσικών επιστημών. Μια σημαντική κλάση εφαρμογών, στην οποία τα παράλληλα περιβάλλοντα υπολογισμού παρέχουν δυνατότητες και προκλήσεις ανάπτυξης, είναι αυτή που σχετίζεται με την επίλυση των μερικών διαφορικών εξισώσεων.

1.2 Αντικείμενο διπλωματικής

Στη δεδομένη διπλωματική εργασία, από το ευρύ φάσμα επίλυσης μερικών διαφορικών εξισώσεων, μας απασχολεί ένα υποσύνολο. Στην εργασία αυτή, μας απασχολεί το υποσύνολο των γραμμικών, μερικών διαφορικών εξισώσεων, δευτέρας τάξης. Ένα χαρακτηριστικό παράδειγμα αυτής της μορφής, είναι η διάδοση της θερμότητας σε αγώγιμα υλικά.

Πιο συγκεκριμένα, μελετάμε την τάξη διαφορικών εξισώσεων που περιγράφονται από τον τρόπο με τον οποίο διαδίδεται η θερμότητα στα εσωτερικά σημεία ενός αγώγιμου υλικού, από μία δεδομένη χρονική στιγμή, όπου η θερμοκρασία του είναι γνωστή, και ύστερα. Η θερμοκρασία στα άκρα του σώματος, κατά το χρονικό διάστημα μελέτης, παραμένει γνωστή. Αν μοντελοποιήσουμε αυτό το πρόβλημα και το γενικεύσουμε, γίνεται αντιληπτό ότι ενδιαφερόμαστε για μερικές διαφορικές εξισώσεις που αποτελούν προβλήματα συνοριακών και αρχικών τιμών.

Στην εργασία μας, χρησιμοποιούμε ως μέθοδο επίλυσης, την Μέθοδο των Γραμμών, που αποτελεί ένα κλασικό αλγόριθμο επίλυσης γραμμικών μερικών διαφορικών εξισώσεων δευτέρας τάξης και έχει πολλές εφαρμογές. Ο αλγόριθμος αυτός, διακριτοποιεί όλες τις διαστάσεις (ανεξάρτητες μεταβλητές), εκτός μίας. Στην υλοποίηση που πραγματοποιούμε, επιλύουμε το πρόβλημα σε δύο διαστάσεις. Κάνοντας υπέρθεση των σημείων αυτών στο χρόνο, σχηματίζονται γραμμές, από όπου και πήρε το όνομά της η μέθοδος.

Μια εφαρμογή της Μεθόδου των Γραμμών, που έχει αναπτυχθεί, είναι η Mol-C. Η Mol-C, εφαρμόζει τον προσαρμοστικό αλγόριθμο της Μεθόδου των Γραμμών, υλοποιημένο σε γλώσσα προγραμματισμού C. Από την περιγραφή του αλγορίθμου, που κάναμε προηγουμένως, γίνεται αντιληπτό ότι η Mol-C μπορεί να προσαρμοστεί σε παράλληλα

υπολογιστικά συστήματα. Αυτό συμβαίνει καθώς επιτρέπει τόσο τον τεμαχισμό, σχηματίζοντας επιμέρους γραμμές ως μια ξεχωριστή ομάδα-διεργασία, όσο και την αντιστοίχιση των διεργασιών αυτών σε ξεχωριστούς υπολογιστές, οι οποίοι επικοινωνούν μεταξύ τους, ανταλλάσσοντας δεδομένα. Παρόλα αυτά, ο τεμαχισμός και η αντιστοίχιση του αρχικού προβλήματος χρειάζονται προσοχή, διότι μπορούν εύκολα να οδηγήσουν σε λάθος αποτελέσματα και μείωση της απόδοσης του αλγορίθμου.

Επόμενο ζήτημα που αντιμετωπίζουμε είναι η επιλογή του παράλληλου υπολογιστικού συστήματος. Σαν χαρακτηριστικό του, πρέπει να παρέχει ευκολία στην επικοινωνία μεταξύ των διαθέσιμων, υπολογιστικών του πόρων. Ένα τέτοιο περιβάλλον, στο οποίο έχουμε άμεση πρόσβασή και που μας προσφέρει δυνατότητες εκτέλεσης της εφαρμογής που αναπτύσσουμε, είναι το cluster του τμήματος, Centaurus.

1.2.1 Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Μελέτη των προβλημάτων γραμμικών, μερικών διαφορικών εξισώσεων δευτέρας τάξης.
2. Ανάλυση της Μεθόδου των Γραμμών, ως μέθοδος επίλυσης μερικών διαφορικών εξισώσεων.
3. Παρουσίαση της Mol-C εφαρμογή της Μεθόδου των Γραμμών.
4. Ανάπτυξη μιας νέας παράλληλης εφαρμογής της Μεθόδου των Γραμμών, που βασίζεται στην Mol-C εφαρμογή.
5. Ενσωμάτωση της εφαρμογής σε παράλληλο υπολογιστικό περιβάλλον, μορφής cluster.
6. Αξιολόγηση των αποτελεσμάτων της παράλληλης εκτέλεσης.

1.3 Οργάνωση κειμένου

Στην επόμενη ενότητα παραθέτουμε τις διαφορικές εξισώσεις, που αποτελούν το γνωστικό αντικείμενό μας και παρουσιάζουμε την επίλυσή τους, μέσω της Μεθόδου των Γραμμών. Στο Κεφάλαιο 3 κάνουμε πλήρη παρουσίαση της παράλληλης εφαρμογής που αναπτύσσουμε, παρουσιάζοντας τόσο κομμάτια της εφαρμογής που έχουν αλγοριθμικό ενδιαφέρον, όσο και τα εργαλεία που χρησιμοποιήσαμε. Στο τέταρτο και τελευταίο κεφάλαιο ανάπτυξης,

καταλήγουμε με την αξιολόγηση των αποτελεσμάτων της εφαρμογής μας και με τις μελλοντικές επεκτάσεις που επιδέχεται. Τέλος, στο παράρτημα της αναφοράς μας, περιγράφουμε έναν οδηγό εγκατάστασης εργαλείων, τα οποία είναι απαραίτητα για την εκτέλεση της εφαρμογής μας.

2

Η Μέθοδος των Γραμμών για την Αριθμητική Επίλυση ΜΔΕ

Πολλά σημαντικά προβλήματα στην τεχνολογία και στις φυσικές επιστήμες , όταν μοντελοποιούνται, απαιτούν τον προσδιορισμό μιας συνάρτησης, που ικανοποιεί κάποια εξίσωση, η οποία περιέχει μία ή περισσότερες παραγώγους της άγνωστης αυτής συνάρτησης. Τέτοιες συναρτήσεις ονομάζονται, **διαφορικές εξισώσεις**. Ένας τρόπος για να κατηγοριοποιήσουμε τις διαφορικές εξισώσεις, είναι μέσω των ανεξάρτητων μεταβλητών από τις οποίες εξαρτάται η άγνωστη συνάρτηση.

Στις περιπτώσεις που η άγνωστη συνάρτηση εξαρτάται από μία μεταβλητή, τότε έχουμε να κάνουμε με **συνήθεις διαφορικές εξισώσεις**, καθώς στη διαφορική εξίσωση περιέχονται μόνο συνήθεις παράγωγοι. Παράλληλα, όταν στη διαφορική εξίσωση υπάρχουν μερικές παράγωγοι, δύο ή περισσοτέρων ανεξάρτητων μεταβλητών, τότε η διαφορική εξίσωση ονομάζεται **μερική διαφορική εξίσωση**.

Μια μερική διαφορική εξίσωση αποτελείται από τη σχέση μίας άγνωστης συνάρτησης δύο ή περισσοτέρων ανεξάρτητων μεταβλητών και των μερικών τους παραγώγων. Οι μερικές διαφορικές εξισώσεις χρησιμοποιούνται για να διαμορφώσουν και να οδηγήσουν στη λύση προβλημάτων, με πολλές διαφορετικές μεταβλητές. Τέτοια προβλήματα είναι η διάδοση του

ήχου ή της θερμότητας, η ροή ρευστού, η ελαστικότητα όπως και προβλήματα ηλεκτροστατικής και ηλεκτροδυναμικής. Το ενδιαφέρον σε αυτή την κατηγορία διαφορικών εξισώσεων είναι ότι, διαφορετικά φυσικά φαινόμενα μπορούν να έχουν ταυτόσημες μαθηματικές διαπιστώσεις και συνεπώς να διέπονται από τις ίδιες δυναμικές.

ΤΥΠΟΙ ΜΕΡΙΚΩΝ ΔΙΑΦΟΡΙΚΩΝ ΕΞΙΣΩΣΕΩΝ:

- Εξίσωση δυναμικού ή εξίσωση LAPLACE:

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = 0 \text{ ή } u_{xx} + u_{yy} = 0$$

- Εξίσωση διάχυσης ή διάδοσης θερμότητας

$$a^2 \frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t} \qquad \text{ή} \quad a^2 u_{xx} = u_t$$

- Κυματική εξίσωση

$$a^2 \frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial^2 u(x,t)}{\partial t^2} \qquad \text{ή} \quad a^2 u_{xx} = u_{tt}$$

- Σφαιρικά κύματα

$$\frac{\partial^2 u(r,t)}{\partial t^2} = c^2 [\frac{\partial^2 u(r,t)}{\partial r^2} + \frac{2}{r}\frac{\partial u(r,t)}{\partial r}] \text{ ή } u_{tt} = c^2 [u_{rr} + \frac{2}{r} u_r]$$

- Εξίσωση Euler-Tricomi

$$\frac{\partial^2 u(x,t)}{\partial x^2} = x \frac{\partial^2 u(x,t)}{\partial t^2} \text{ ή } u_{xx} = x u_{tt}$$

2.1 Προβλήματα ΜΔΕ

Οι μερικές διαφορικές εξισώσεις είναι ένα ευρύ μαθηματικό πεδίο. Στην εν λόγω διπλωματική εργασία ασχολούμαστε με τις **γραμμικές διαφορικές εξισώσεις, δευτέρας τάξης**. Ως **τάξη** μιας μερικής διαφορικής εξίσωσης αναφέρουμε το μέγιστο βαθμό μερικής παραγώγου οποιασδήποτε ανεξάρτητης μεταβλητής, που εμφανίζεται στην εξίσωση ([1]). Την έννοια της **γραμμικότητας**, τη συσχετίζουμε με το γεγονός ότι η άγνωστη συνάρτηση u, εμφανίζεται παντού, μόνο στην πρώτη της δύναμη και όχι σε σύνθεση με άλλη συνάρτηση. Συνεπώς, στην περίπτωσή μας οι μερικές διαφορικές εξισώσεις είναι της μορφής :

$$u_t = f * u_{xx} \text{ ή } \frac{\partial u(x,t)}{\partial t} = f \frac{\partial^2 u(x,t)}{\partial x^2} \qquad (2.1)$$

Όπου

- x,η ανεξάρτητη μεταβλητή που αντιστοιχεί στο χώρο με $x_0 \le x \le x_N$
- t,η ανεξάρτητη μεταβλητή που αντιστοιχεί στο χρόνο με $t_0 \le t \le t_K$
- u(x,t),η άγνωστη συνάρτηση δύο ανεξαρτήτων μεταβλητών του χρόνου και του χώρου, την μορφή της οποίας αναζητούμε.
- f μπορεί να είναι:
    - μια σταθερά (όπως συμβαίνει στην διαφορική εξίσωση της μετάδοσης θερμότητας),
    - μια συνάρτηση του χώρου, του χρόνου ή και των δύο ταυτόχρονα.

Εμείς ασχολούμαστε με την τελευταία περίπτωση, όπου η f είναι συνάρτηση τόσο του χρόνου όσο και του χώρου.

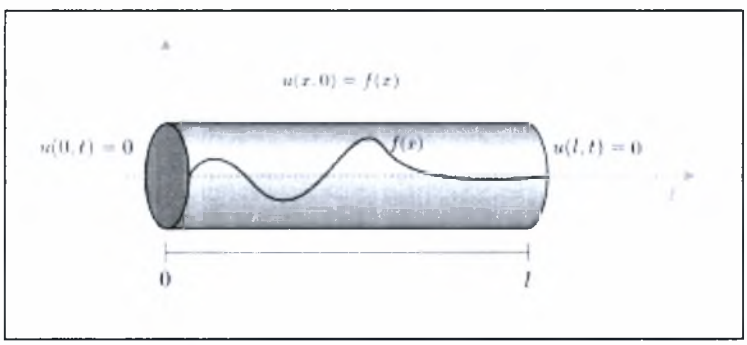
Για να βρούμε τη λύση μιας μερικής διαφορικής εξίσωσης, κάνουμε χρήση των πληροφοριών που έχουμε, τόσο για την άγνωστη αυτή συνάρτηση u(x,t) όσο και για τις μερικές της παραγώγους. Οι μερικές διαφορικές εξισώσεις αποτελούν προβλήματα **αρχικών τιμών (Cauchy)**, με αποτέλεσμα για να τις επιλύσουμε, πρέπει να γνωρίζουμε τις τιμές της άγνωστης συνάρτησης u(x,t), την χρονική στιγμή $t_0=0$ για κάθε τιμή του x,με $x_0 \le x \le x_N$.

Παράλληλα, οι μερικές διαφορικές εξισώσεις αποτελούν προβλήματα **συνοριακών τιμών**. Ένα πρόβλημα **συνοριακών τιμών** είναι μια διαφορική εξίσωση μαζί με ένα σύνολο από επιπρόσθετους περιορισμούς, που καλούνται **συνοριακές συνθήκες**, Η λύση σε ένα τέτοιο πρόβλημα είναι μια συνάρτηση, η οποία ικανοποιεί τις δοσμένες συνοριακές συνθήκες. Για να επιλύσουμε ένα πρόβλημα συνοριακών τιμών πρέπει να είναι **καλά ορισμένο**. Αυτό σημαίνει ότι πρέπει να δίνουμε τα δεδομένα εισόδου του προβλήματος (αρχικές τιμές και συνοριακές συνθήκες) , για να βρούμε τη λύση, η οποία εξαρτάται κάθε χρονική στιγμή από τα δεδομένα αυτά. Συνεπώς, πρέπει για κάθε χρονική στιγμή t, $t_0 \le t \le t_K$ να γνωρίζουμε την τιμή της u(x,t) στις συνοριακές συνθήκες της, $u(x_0,t)$ και $u(x_N,t)$.

Εφαρμογή αυτής της μορφής διαφορικών εξισώσεων αποτελεί το φαινόμενο της διάδοσης της θερμότητας σε αγώγιμα υλικά. Αυτό αποτελεί και ένα καλό παράδειγμα για να κατανοήσουμε τα όσα αναφέραμε νωρίτερα, δηλαδή ότι οι μερικές διαφορικές εξισώσεις

συνιστούν καλά ορισμένα προβλήματα, έχοντας ως δεδομένο τις αρχικές τιμές και τις συνοριακές συνθήκες.

Υποθέτουμε ότι έχουμε μια ευθύγραμμη ράβδο ομοιόμορφης διατομής και ομογενούς υλικού([2]). Ταυτόχρονα θεωρούμε ότι οι πλευρές της ράβδου είναι καλά μονωμένες , ώστε να μην έχουμε μεταφορά θερμότητας δια μέσου αυτών και ότι οι διαστάσεις της διατομής είναι μικρές, ώστε η θερμοκρασία να θεωρείται σταθερή σε οποιοδήποτε σημείο της διατομής. Έτσι, η εύρεση της θερμοκρασίας σε κάθε σημείο της ράβδου (χωρική ανεξάρτητη μεταβλητή), για κάθε χρονική στιγμή, γνωρίζοντας την θερμοκρασία σε κάθε σημείο της για τη χρονική στιγμή που ορίζουμε ως t=0, καθώς και τις τιμές της θερμοκρασίας στα άκρα της x=0 T_0=K και x=L, T_L=M, είναι η λύση της μερικής διαφορικής εξίσωσης της μορφής (2.1) που ικανοποιεί αυτές τις συνθήκες και έχει f = α (συντελεστής θερμικής διάχυσης, σταθερά). Στο Σχήμα 2.1 αναπαριστούμε την αρχικοποίηση του προβλήματος κατανομής της θερμότητας.



Σχήμα 2.1 Ευθύγραμμη ράβδος μικρής διατομής, ομογενούς υλικού, με μονωμένα άκρα

Η λύση μιας διαφορικής εξίσωσης, όπως αναφέραμε και προηγουμένως, είναι μια συνάρτηση που προσδιορίζει τις εξαρτημένες μεταβλητές συναρτήσει των ανεξάρτητων, στην περίπτωση μας την u(x,t) βάσει των x και t. Ουσιαστικά, καλούμαστε να βρούμε μια συνάρτηση η οποία όταν αντικατασταθεί στην διαφορική εξίσωση, να ικανοποιούνται ταυτόχρονα όλες οι απαιτούμενες συνθήκες. Υπάρχουν δύο τύποι λύσεων([8]):

1. Αν η συνάρτηση είναι μια πραγματική μαθηματική συνάρτηση, τότε η λύση καλείται **αναλυτική**. Η αναλυτική λύση είναι ακριβής αλλά και είναι δύσκολο να βρεθεί για κάθε PDE πρόβλημα.
2. Αν η λύση είναι αριθμητικού τύπου, τότε καλείται **αριθμητική**. Ουσιαστικά, η αριθμητική λύση είναι μια αριθμητική εκτίμηση της αναλυτικής λύσης. Από την στιγμή που η αναλυτική λύση είναι γενικά δύσκολη για ρεαλιστικά προβλήματα

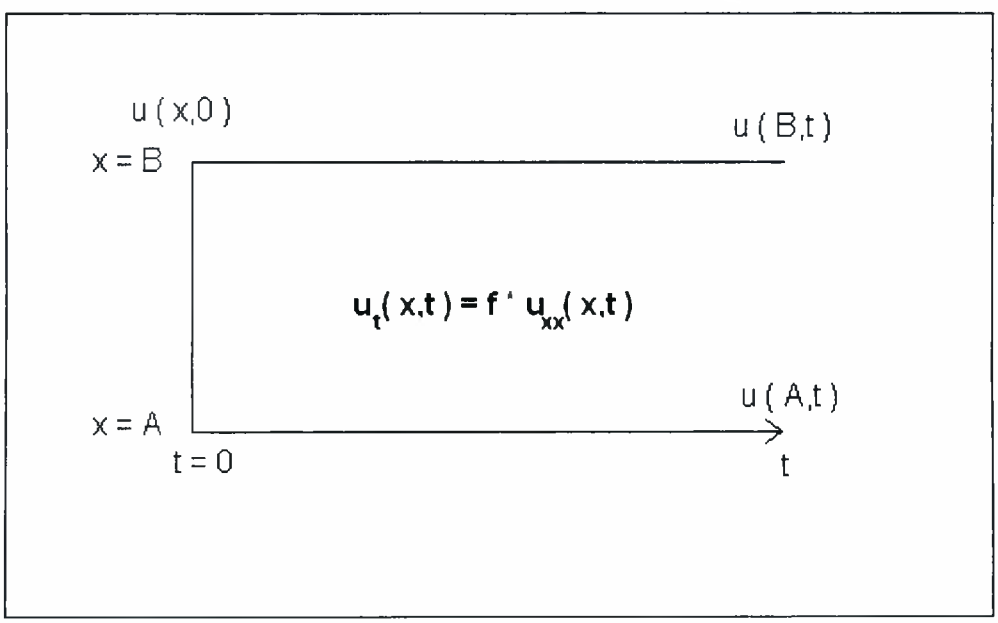
μερικών διαφορικών εξισώσεων, οι αριθμητικές λύσεις αποτελούν μια προσέγγιση τους. Επιδίωξή μας σε αυτή την μορφή λύσης, είναι να ανταποκρίνονται στις αναλυτικές λύσεις με μεγάλη ακρίβεια.

Στη συνέχεια αυτής της ενότητας δίνουμε γενικά στοιχεία της αριθμητικής λύσης των μερικών διαφορικών εξισώσεων, της μορφής (2.1) με τις οποίες θα ασχοληθούμε στην συγκεκριμένη εργασία. Ακόμα, παρουσιάζουμε μια εφαρμογή της μεθόδου των γραμμών με την οποία διαπιστώνουμε την συμπεριφορά της συναρτήσει των παραμέτρων της, καθώς και την μέθοδο επίλυσης που θα χρησιμοποιήσουμε για να λύσουμε τις μερικές διαφορικές εξισώσεις που μας απασχολούν.

2.2 Αριθμητική Επίλυση ΜΔΕ

Οι μερικές διαφορικές εξισώσεις που μας ενδιαφέρουν, αποτελούν εφαρμογές προβλημάτων συνοριακών συνθηκών, όπως αναφέρθηκε νωρίτερα. Αυτό σημαίνει ότι για να τις επιλύσουμε πρέπει να δίνουμε τις συνοριακές τιμές της εξίσωσης, στο πεδίο ορισμού του προβλήματος, καθώς και οι αρχικές τιμές του.

Στο Σχήμα 2.2 που ακολουθεί δίνουμε μια γραφική απεικόνιση του πεδίου επίλυσης της διαφορικής εξίσωσης της μορφής (2.1) που μας ενδιαφέρει. $u_t = f^* u_{xx}$



Σχήμα 2.2. Απεικόνιση του πεδίου ορισμού της μερικής διαφορικής εξίσωσης

Έτσι για να μπορέσουμε να βρούμε μια συνάρτηση u(x,t), η οποία να επαληθεύει την μερική διαφορική εξίσωση, πρέπει να γνωρίζουμε :

- Την αρχική συνθήκη της u(x,t) για t=0
- Τις συνοριακές τιμές της u(x,t) :
    - Την κάτω συνοριακή τιμή : u(A,t)=h(t)
    - Την άνω συνοριακή τιμή : u(B,t)=g(t)

Υπάρχουν αρκετές μέθοδοι που έχουν αναπτυχθεί για την επίλυση των μερικών διαφορικών εξισώσεων. Ονομαστικά αναφέρουμε ορισμένες από αυτές :

- Μέθοδος διαχωρισμού των μεταβλητών
- Μέθοδος αλλαγής μεταβλητής
- Μέθοδος των χαρακτηριστικών
- Μέθοδος Υπέρθεσης της Αρχής
- Σειρές Fourier

Στην διπλωματική αυτή αναπτύσσουμε και χρησιμοποιούμε ως μέθοδο επίλυσης των μερικών διαφορικών εξισώσεων της μορφής (2.1) την **Μέθοδο Των Γραμμών**.

2.3 Η Μέθοδος των Γραμμών

Η μέθοδος των γραμμών είναι μια γενική τεχνική επίλυσης μερικών διαφορικών εξισώσεων (partial differential equations – PDEs), στην οποία διακριτοποιούμε όλες τις διαστάσεις εκτός μίας, και έπειτα ολοκληρώνουμε το διακριτοποιημένο πρόβλημα ως ένα σύστημα από συνήθεις διαφορικές εξισώσεις. Ένα σημαντικό πλεονέκτημα αυτής της μεθόδου είναι ότι κάνουμε χρήση των μεθόδων γενικού σκοπού και των λογισμικών, που ήδη έχουν αναπτυχθεί για την ολοκλήρωση μερικών διαφορικών εξισώσεων . Ακόμα, προτέρημα αυτής της μεθόδου αποτελεί το γεγονός, ότι οι μερικές διαφορικές εξισώσεις στις οποίες απευθύνεται και μας απασχολούν, επωφελούνται από την εφαρμογή τους σε περιβάλλοντα πολλαπλών υπολογιστικών πόρων, όπως το **GRID** ή ένα **cluster** ([4]).

Είναι απαραίτητο το πρόβλημα των μερικών διαφορικών εξισώσεων να είναι καλά ορισμένο ως πρόβλημα αρχικών τιμών (Cauchy) τουλάχιστον ως προς τη μια διάσταση που

διακριτοποιείται. Αυτό συμβαίνει γιατί μετά την διακριτοποίηση, το πρόβλημα ανάγεται σε πρόβλημα επίλυσης συνήθων διαφορικών εξισώσεων, που είναι πρόβλημα αρχικών τιμών.

2.3.1 Βασικός Αλγόριθμος

Η μέθοδος των γραμμών είναι ένας ισχυρός αλγόριθμος μετατροπής μιας παραβολικής μερικής διαφορικής εξίσωσης ([4]) της μορφής

$$u_t(t,x) = f(x,t)u_{xx}(t,x) \qquad (2.3.1)$$

σε ένα σύστημα από συνήθεις διαφορικές εξισώσεις (ODEs). Για την διαφορική εξίσωση (2.3.1) σε σύστημα δύο διαστάσεων έχουμε :

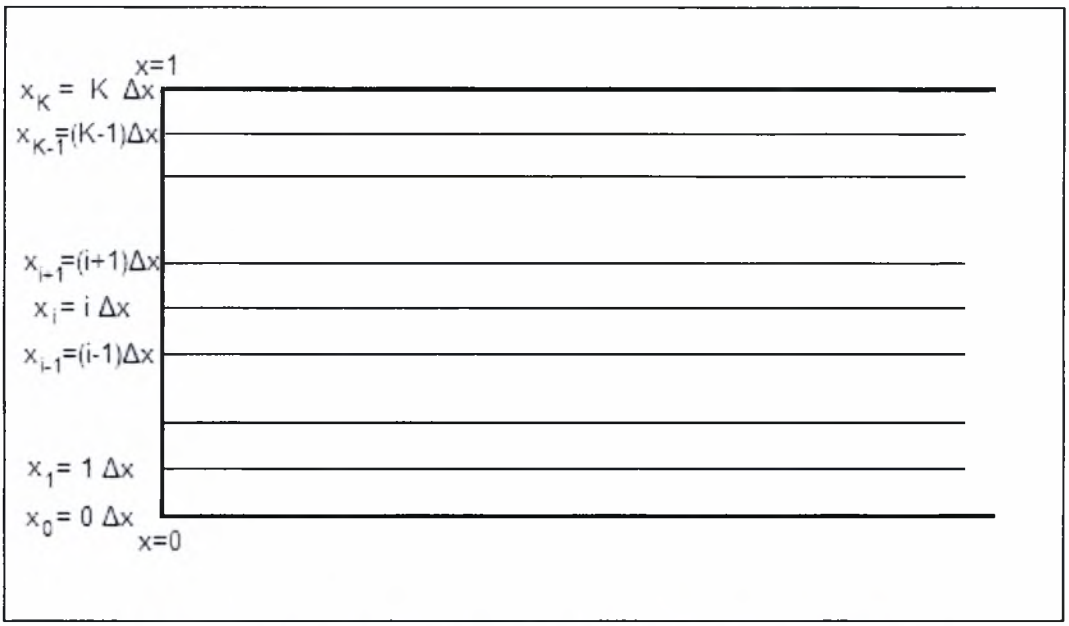
- για t=0, $x_0 \le x \le x_K$
- οι υπολογισμοί ξεκινούν για t=0 και συνεχίζονται σε όλο το διάστημα του χρόνου.

Η αναπαράσταση του δισδιάστατου αυτού πεδίου είναι όμοια με αυτή του Σχήματος 2.2 που παρουσιάσαμε στην προηγούμενη ενότητα.

Τα βήματα του αλγορίθμου της μεθόδου των γραμμών, που εκτελούμε είναι τα ακόλουθα ([5]):

1. Διακριτοποιούμε το διάστημα που ορίζεται η χωρική μεταβλητή x, σε **K σημεία** (**K+1 γραμμές**) , με μεταξύ τους απόσταση Δx. Συνεπώς τα σημεία που δημιουργούμε είναι τα: $\mathbf{x_i=x_0+k*\Delta x}$,όπου **i=0, ..., K**. Αυτό ορίζει K-1 νέες παράλληλες προς τις στο $x=x_0$ και $x=x_K$, γραμμές. Την u(x,t) τη γνωρίζουμε σε κάθε μια από αυτές τις γραμμές για τη χρονική στιγμή t=0, ενώ για κάθε άλλη χρονική στιγμή μας είναι γνωστές μόνο οι συνοριακές τιμές για x_k, για k= 0, K (συνοριακές συνθήκες, Σχήμα 2.3.1). Το Δx δίνεται από την σχέση:

$$\Delta x = \frac{x_K - x_0}{K+1} \qquad (2.3.2)$$



Σχήμα 2.3.1 Διακριτοποίηση της χωρικής ανεξάρτητης μεταβλητής x

2. Στη διακριτοποιημένη διάσταση χρησιμοποιούμε έναν τύπο πεπερασμένων διαφορών (2.3.3) για αντικατάσταση των u_{xx}

$$u_{xx}(t,x_i)=\frac{u(t,x_i+\Delta x)-2u(t,x_i)+u(t,x_i-\Delta x)}{(\Delta x)^2} \qquad (2.3.3)$$

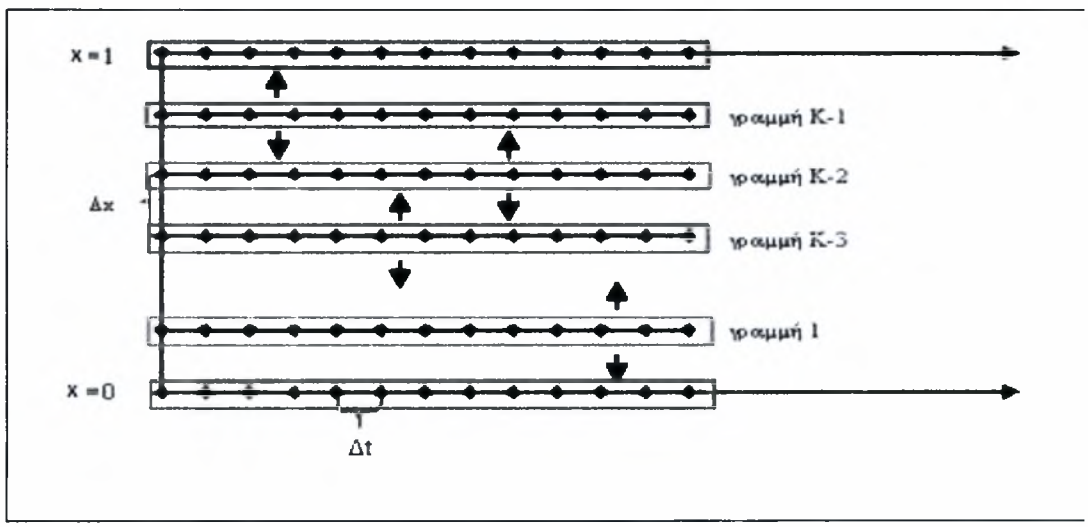
3. Έτσι δημιουργούμε K-1 συνήθεις διαφορικές εξισώσεις, στις εσωτερικές γραμμές. Όλες μαζί αποτελούν ένα Σύστημα Συνήθων Διαφορικών Εξισώσεων, το οποίο καλούμαστε να επιλύσουμε. Για να απλοποιήσουμε την σημειογραφία θέτουμε $u^{(i)}=u(x,t)\big|_{x=x_i}$, με $t\in[0,T]$ και i=1,…,K . Κάνοντας χρήση αυτής της σημειογραφίας, μετατρέπουμε τη σχέση (2.3.3) ως ακολούθως:

$$u_{xx}^{(i)}(t)=\frac{u^{(i+1)}(t)-2u^{(i)}(t)+u^{(i-1)}(t)}{(\Delta x)^2} \qquad (2.3.4)$$

Έτσι αν αντικαταστήσουμε την (2.3.4) στην (2.3.1), λαμβάνουμε το σύστημα συνήθων διαφορικών εξισώσεων :

$$u_t^{(i)}(t)=f(x_i,t)\frac{u^{(i+1)}(t)-2u^{(i)}(t)+u^{(i-1)}(t)}{(\Delta x)^2}+O(\Delta x^2) \quad (2.3.5)$$

Στο επόμενο σχήμα παρουσιάζουμε τη μορφή του προβλήματος όπως γίνεται μετά την διακριτοποίηση (Σχήμα 2.3.2).



Σχήμα 2.3.2 Γραφική αναπαράσταση του συστήματος Συνήθων Διαφορικών Εξισώσεων, στο οποίο ανάγεται το πρόβλημα μετά τη διακριτοποίηση.

4. Στο σύστημα των συνήθων διαφορικών εξισώσεων, που αντιμετωπίζουμε, εφαρμόζουμε έναν πρότυπο ODE επιλυτή, σε κάθε γραμμή. Με αυτόν εκτελούμε υπολογισμούς, στο t=t_j της μορφής

$$u_i^{(j)}(t_j + \Delta t) = \text{ κάποια συνάρτηση πληροφορίας για } t \le t_i$$

Και στην πορεία: (1) κάνουμε ένα βήμα προς τα μπροστά μεγέθους Δt, (2) ανταλλάσσουμε πληροφορία με τις k+1 και k-1 γραμμές και (3) επαναλαμβάνουμε τα προηγούμενα βήματα.

Τα παραπάνω βήματα τα εκτελούμε με γνώμονα την επίτευξη (**απαιτήσεις προβλήματος**):

- **Συγκεκριμένου ρυθμού προόδου**, όπου ρυθμός προόδου είναι ο ελάχιστος ρυθμός στον οποίο οι υπολογισμοί πρέπει να παράγουν λύσεις. Για παράδειγμα, αν υποθέσουμε ότι ο καθορισμένος ρυθμός προόδου είναι 30 λύσεις ανά δευτερόλεπτο, πρέπει να παράγονται 30 στιγμιότυπα λύσεων σε κάθε δευτερόλεπτο.

- **Ζητούμενης ανοχής**, η οποία αφορά το μέγιστο σφάλμα υπολογισμών, κατά το οποίο εξακολουθεί να θεωρείται αποδεκτή, η λύση του κάθε υπολογιστικού βήματος.

Οι παράμετροι που καθορίζουν το πρόβλημα της επίλυσης της μερικής διαφορικής εξίσωσης είναι οι ακόλουθοι:

- **Ρυθμός Προόδου**: ο ελάχιστος ρυθμός στον οποίο παράγονται λύσεις
- **α** : η παράμετρος του sin(αt)
- **b**: η παράμετρος του cos(bx)
- **Ακρίβεια**: η απαιτούμενη ανοχή κατά την επίλυση της μερικής διαφορικής εξίσωσης
- **K**: το πλήθος των διαστημάτων της μεθόδου
- **Επιλογή μεθόδου ODE επιλυτή** που χρησιμοποιείται μεταξύ των γραμμών
- **Τάξη της μεθόδου του ODE επιλυτή**
- **Το πλήθος των ισότιμών ομάδων** στις οποίες χωρίζονται ο γραμμές του προβλήματος. Είναι ίσο με το πλήθος των διαθέσιμων επεξεργαστών που χρησιμοποιούνται στην παράλληλη υλοποίηση. Δεν μας απασχολεί στο σημείο αυτό.

Αυτός είναι ο βασικός αλγόριθμος της μεθόδου των γραμμών. Εδώ πρέπει να αναφέρουμε ότι αυτή η μέθοδος είναι προσαρμοστική με αποτέλεσμα μετά από κάθε υπολογιστικό βήμα να εκτελείται έλεγχος αν ικανοποιούνται οι προδιαγραφές του ρυθμού προόδου και της ανοχής. Στην επόμενη ενότητα αναπτύσσουμε την προσαρμοστικότητα της μεθόδου των γραμμών.

Πάνω σε αυτόν το αλγόριθμο έχουν στηριχθεί πολλές εφαρμογές της μεθόδου των γραμμών. Εμείς υλοποιήσαμε, για τις ανάγκες κατανόησης της συμπεριφοράς της Μεθόδου των Γραμμών μια μη προσαρμοστική υλοποίηση σε Matlab.

2.3.2 Μη προσαρμοστική επίλυση, Matlab

Για τις ανάγκες κατανόησης της συμπεριφοράς της μεθόδου των γραμμών, αναπτύσσουμε μια μη προσαρμοστική εφαρμογή της. Σε αυτή την υλοποίηση, δίνουμε τις παραμέτρους του προβλήματος, το επιλύουμε, χρησιμοποιώντας ως επιλυτή των συνήθων διαφορικών εξισώσεων, τον πρότυπο ODE επιλυτή του Matlab : ode113 που αντιστοιχεί στην μέθοδο Adams-Moulton, και παίρνουμε τις γραφικές παραστάσεις.

Σε αυτή την υλοποίηση (φάκελος pde_1D) , όπως και στις επόμενες, χρησιμοποιούμε ως πρότυπη λύση του προβλήματος την:

$$u(t,x) = \cos(bx) * [2 + \sin(at)] \qquad (2.3.2.1)$$

Με αποτέλεσμα να δίνουμε στο πρόβλημα την ακόλουθη μορφή:

$$u_t(x,t) = -[\frac{a * \cos(at)}{b^2 * (2 + \sin(at))}]u_{xx}(x,t) \qquad (2.3.3.2)$$

Μόλις ξεκινάμε την εφαρμογή (αρχείο pdesolve.m) σε περιβάλλον Matlab, εκτελούμε τις ακόλουθες εισαγωγές:

- Τιμή του κάτω ορίου της χωρικής μεταβλητής x, x_0.
- Τιμή του άνω ορίου της x, x_K.
- Τιμή χρονικής παραμέτρου α
- Τιμή χωρικής παραμέτρου b
- Πλήθος διαστημάτων διακριτοποίησης, από το οποίο υπολογίζουμε το Δx.
- Διάστημα διεξαγωγής υπολογισμών, TSPAN.
- Τιμή της ανοχής:, relative
- Τιμή άνω ορίου, κάθε βήματος step.

Για να χρησιμοποιήσουμε στους υπολογισμούς μας, τον πρότυπο ODE επιλυτή Adams-Moulton εκτελούμε την ακόλουθη εντολή:

[T Y] = ode113(@myode, TSPAN, IC, options)

Όπου, τα ορίσματα της συνάρτησης ode113 είναι :

- myode (αρχείο myode.m) : η συνήθης διαφορική εξίσωση που αντιστοιχεί στην διακριτοποιημένη μερική διαφορική εξίσωση (2.3.3.2):

$$u_t^{(i)}(t) = -[\frac{a * \cos(at)}{b^2 * (2 + \sin(at))}] * \frac{u^{(i+1)}(t) - 2u^{(i)}(t) + u^{(i-1)}(t)}{(\Delta x)^2} \qquad (2.3.3.3)$$

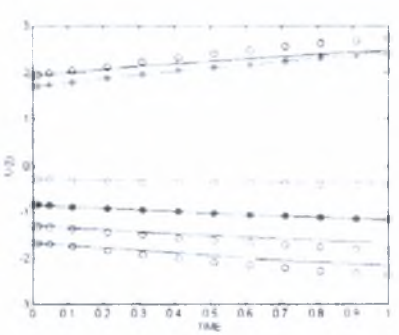
- TSPAN: το διάστημα διεξαγωγής υπολογισμών
- IC (initial conditions): Πίνακας που διατηρεί τις αρχικές τιμές του προβλήματος τη χρονική στιγμή t_0=0
- options: σύνολο από επιλογές της ode113 μεθόδου. Εμείς περνάμε ως επιλογές τις παραμέτρους relative και step που ορίσαμε νωρίτερα, μέσω της δήλωσης:

- options= odeset(' RelTol', relative, 'MaxStep', step)

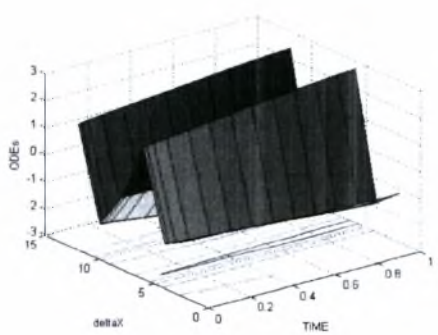
  η συνάρτηση odeset είναι προκαθορισμένη από το Matlab να αντιστοιχεί τις παραμέτρους του προβλήματος που βρίσκονται μέσα στα '', με τις μεταβλητές που τις ακολουθούν.

- Η ode113 επιστρέφει :
  - T: τον μονοδιάστατο πίνακα των χρονικών σημείων του διαστήματος TSPAN στα οποία εκτέλεσε ο Adams-Moulton επιλυτής, υπολογισμούς.
  - Y: τον πίνακα δύο διαστάσεων, που αντιστοιχεί στους υπολογισμούς που εκτέλεσε ο επιλυτής στις γραμμές, τις διάφορες χρονικές στιγμές. Η στήλες αναφέρονται στις χρονικές στιγμές στις οποίες έγιναν υπολογισμοί και οι γραμμές του πίνακα, στα σημεία που διακριτοποιήθηκε το πεδίο του προβλήματος.

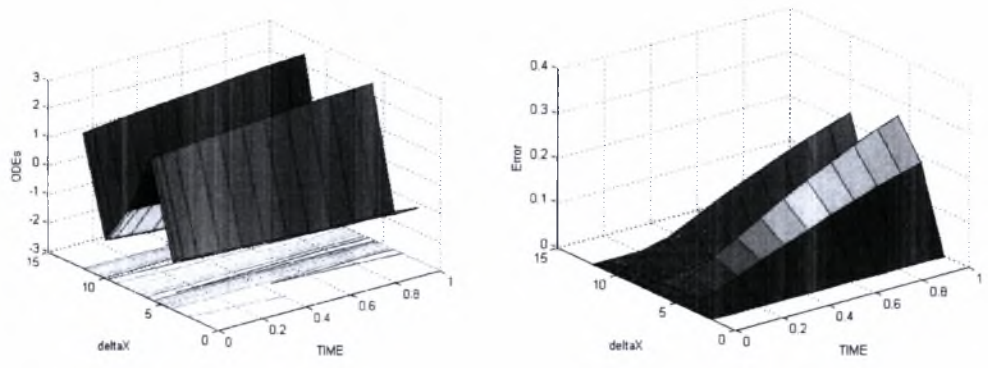
Μετά από την προηγούμενη περιγραφή, εμφανίζουμε τις γραφικές παραστάσεις, Σχήμα 2.3.2.1. Αρχικά στο ίδιο σχήμα δύο διαστάσεων, παρουσιάζουμε τη σύγκριση των υπολογισμένων τιμών, με τις πραγματικές (Σχήμα 2.3.2.1.a., όπου με 'o' παριστάνουμε τις πραγματικές). Ακολουθεί η τρισδιάστατη γραφική απεικόνιση της PDE (Σχήμα 2.3.2.1.**b**.), όπως την λύσαμε με την Adams – Moulton στις σύνηθεις διαφορικές εξισώσεις που δημιουργούμε με την διακριτοποίηση, και έπειτα, η αντίστοιχη των πραγματικών λύσεων (Σχήμα 2.3.2.1.c.). Τέλος, (Σχήμα 2.3.2.1.d.) οι τρισδιάστατη διαφορά των δύο λύσεων.



a.



b.



c. **d.**

Σχήμα 2.3.2.1 Γραφικές παραστάσεις pdesolve Matlab εφαρμογής

Οι γραφικές παραστάσεις, που παρουσιάσαμε, είναι αυτές που αντιστοιχούν στις παρακάτω τιμές των παραμέτρων του προβλήματος:

- Κάτω όριο, $x_0 = 2$
- Άνω όριο, $x_K = 12$
- Πλήθος διαστημάτων 5
- TSPAN = [0 1]
- a=1
- b=1
- RelTol = default (e^{-3}, δηλ. 0.001)
- MaxStep = default (το ένα δέκατο του διαστήματος TSPAN, δηλ. 0.1)

Αυτή είναι μια υλοποίηση περιορισμένων της μεθόδου των γραμμών, περιορισμένων δυνατοτήτων, γιατί δεν εκτελεί προσαρμογή μετά τις μετρήσεις της. Για αυτό, στη διπλωματική εργασία κάνουμε χρήση της προσαρμοστικής εφαρμογής που έχει αναπτυχθεί σε C και ονομάζεται MOL-C ([5,6,7]).

2.3.3 Παρουσίαση της MOL – C εφαρμογής

Για τις ανάγκες της υλοποίησης της εν λόγω διπλωματικής εργασίας χρησιμοποιήσαμε το MOL-C πρόγραμμα, που ήδη έχει αναπτυχθεί ([5]). Αυτή η εφαρμογή της MOL αποτελεί τον πυρήνα της παράλληλης υλοποίησης που παρουσιάζουμε αργότερα και έχει αναπτυχθεί σε γλώσσα προγραμματισμού C. Το πρόγραμμα βασίζεται σε παλαιότερη εφαρμογή της MOL

([6,7]), με την καίρια διαφορά ότι απλοποιείται σε συστατικά (components) τα οποία απαιτούνται για επιτάχυνση των αποτελεσμάτων. Σε αυτό οφείλεται η μείωση των γραμμών του προγράμματος από τις 20000 με 30000 γραμμές σε 1000 γραμμές.

Για την ανάπτυξη της συγκεκριμένης εφαρμογής, κάναμε χρήση της μερικής διαφορικής εξίσωσης, που έχει πραγματική λύση την:

$$u(t,x) = \cos(bx) * [2 + \sin(at)] \qquad (2.3.3.1)$$

Συνεπώς η μερική διαφορική εξίσωση παίρνει την μορφή,

$$u_t(x,t) = -[\frac{a * \cos(at)}{b^2 * (2 + \sin(at))}]u_{xx}(x,t) \qquad (2.3.3.2)$$

Όπου α χρονικός παράγοντας, b χωρικός παράγοντας , που για την συγκεκριμένη υλοποίηση τις θεωρούμε σταθερές.

Στη παρουσίαση του βασικού αλγορίθμου (2.3.1), παρουσιάσαμε και τις παραμέτρους, τις οποίες πρέπει να καθορίσουμε για την επίλυση της μερικής διαφορικής εξίσωσης (2.3.3.2), με τη μέθοδο των γραμμών. Για τον προσδιορισμό αυτών των παραμέτρων όπως και για τον καθορισμό των αρχικών τιμών και των συνοριακών συνθηκών, χρησιμοποιούμε ένα αρχείο εισόδου, το **params.txt**. Αυτό το αρχείο, το θέτουμε ως είσοδο της συνάρτησης εκκίνησης της εφαρμογής **main**, που βρίσκεται στο αρχείο **test_method_of_lines_1d.c**, διαβάζεται, και τα δεδομένα που εισάγει τα αποθηκεύουμε στις αντίστοιχες μεταβλητές ενός αντικειμένου **inputs** της δομής δεδομένων **inputStruct**. Στο επόμενο βήμα της main, με την χρήση αυτού του αντικειμένου κάνουμε την αρχικοποίηση της κύριας δομής της εφαρμογής, μέσω της συνάρτησης initialize_mol() που αναπτύσσουμε στο αρχείο **class_method_of_lines_1d.c**.

Η κεντρική δομή δεδομένων που χρησιμοποιούμε στη MOL εφαρμογή είναι η **molStruct**, η οποία αποτελείται από τύπους δεδομένων :

- Δομή **innerSolution**, στα μέλη της οποίας αποθηκεύουμε το σύνολο των εσωτερικών μεταβλητών, που υπολογίζονται από τον ODE επιλυτή. Ενδεικτικά αναφέρουμε ορισμένα που θα μας απασχολήσουν παρακάτω όπως:
    - **pointerToCurrentTime**: ένας δείκτης προς την τρέχουσα χρονική στιγμή υπολογισμών.
    - **deltaX** και **deltaTime**: οι τιμές των παραμέτρων Δx και Δt.

- **solutionTable**: ένας μονοδιάστατος πίνακας μεγέθους αντίστοιχου με το πλήθος των υπολογιστικών βημάτων και το μέγιστο επιτρεπτό πλήθος γραμμών ανά βήμα.
- **previousSolutionTable**: τις τιμές του προηγούμενου υπολογιστικού βήματος

- Δομή **ropStruct**, στην οποία περιέχονται μονάδες που αποθηκεύουν πληροφορία σχετικά με το πόσο διήρκησε ο υπολογισμός του τελευταίου υπολογιστικού βήματος t (**tSnap**), πόσα βήματα χρειάζονται για το επόμενο στιγμιότυπο(**numberOfSnapShots**), και την απαιτούμενη ανοχή (**tolerance**). Αυτή την πληροφορία τη χρησιμοποιούμε για να καθορίσουμε πότε ο ζητούμενος ρυθμός προόδου έχει επιτευχτεί και πότε η ακρίβεια των υπολογισμών είναι ικανοποιητική, συγκρινόμενη με την καθορισμένη ανοχή.
- Δομή **solverStruct**. Σε αυτή τη δομή διατηρούμε την πληροφορία για:
  - Το υπολογισμένο λάθος, **error**.
  - Την τάξη του ODE επιλυτή, **solverOrder**.
  - Μια μεταβλητή αλλαγής της τάξης του ODE επιλυτή, **numChangeOfOrder**
  - Έναν πίνακα που διατηρεί την τάξη του επιλυτή κατά την πορεία των υπολογισμών.

Ολόκληρη η εφαρμογή που έχει αναπτυχθεί, είναι δομημένη σε συνιστώσες (components) που αποτελούν ξεχωριστά αρχεία, ώστε να είναι εύκολη η διαχείριση και η μετατροπή των συναρτήσεων τους. Στην ακόλουθη ανάλυση, κάνουμε μία σύντομη περιγραφή των συνιστωσών αυτών και των συναρτήσεων που τις απαρτίζουν, για να γίνει πιο αντιληπτή η μορφή της C- Mol εφαρμογής:

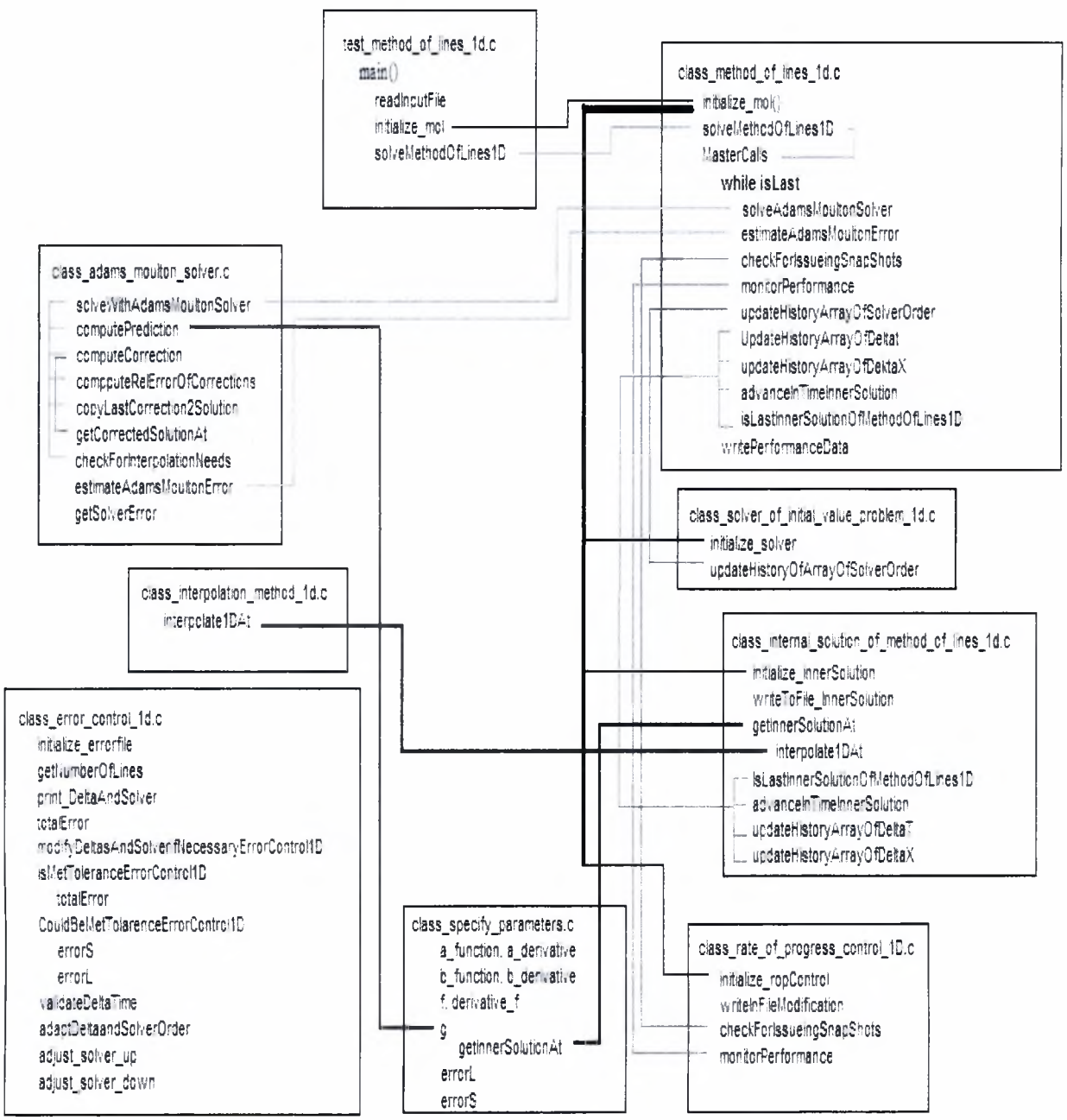
- Αρχείο test_method_of_lines_1d.c: Συναρτήσεις:
  - main()
  - κλήση της readInputFile()
  - κλήση της initialize_mol()
  - κλήση της solveMethodOfLines1D()
- Αρχείο class_method_of_lines_1d.c: Συναρτήσεις:
  - initialize_mol() : η συνάρτηση αυτή πραγματοποιεί την αρχικοποίηση του στιγμιότυπου mol της δομής δεδομένων molStruct, και συνεπώς την αρχικοποίηση του στιγμιότυπου ropControl της δομής ropStruct, του solver της δομή solverStruct και του solution της δομής innerSolution.

- solveMethodOfLines1D: Η κύρια λογική της MOL επίλυσης, πραγματοποιείται στη MasterCalls που καλείται από αυτή τη ρουτίνα.
- MasterCalls: κύρια ρουτίνα της Mol-C εφαρμογής.
- gettime: συνάρτηση που χρησιμεύει για στην εκτίμηση του χρόνου.

- class_internal_solution_of_method_of_lines_1d.c: Στο οποίο ορίζονται οι ακόλουθες συναρτήσεις:
  - initialize_innerSolution: που αρχικοποιεί το στιγμιότυπο της δομής innerSolution, solution
  - writeToFile_InnerSolution: καταγράφει τις ενδιάμεσες λύσεις του προβλήματος
  - getInnerSolutionAt: είναι η κύρια λειτουργία αυτού του αρχείου. Αυτή η συνάρτηση χρησιμοποιείται για να αποκτηθεί η λύση μίας συγκεκριμένης γραμμής σε ένα καθορισμένο χρονικό σημείο. Αν η λύση δεν έχει υπολογιστεί, τότε η συνάρτηση αυτή επιστρέφει ως λύση την παρεμβολή σε εκείνο το σημείο των υπολογισμών, καλώντας την συνάρτηση interpolate1DAt.
  - IsLastInnerSolutionOfMethodOfLines1D: ελέγχει αν το χρονικό σημείο των υπολογισμών είναι το τελικό, όπως ορίζεται από το αρχείο εισόδου
  - updateHistoryArrayOfDeltaT, updateHistoryArrayOfDeltaX: μέθοδοι που προσθέτουν τις νέες τιμές των Δx και Δt
  - advanceInTimeInnerSolution:
- class_rate_of_progress_control_1D.c: Συναρτήσεις που ορίζει:
  - initialize_ropControl: αρχικοποίηση του στιγμιότυπου ropControl της δομής ropStruct
  - writeInFileModification
  - checkForIssueingSnapShots
  - monitorPerformance: μέθοδος που ελέγχει την απόδοση της εφαρμογής
- class_adams_moulton_solver.c : αυτό το αρχείο υλοποιεί τον Adams-Moulton επιλυτή για την επίλυση των ODEs. Δίνοντας ως παραμέτρους την τάξη της μεθόδου, την τρέχουσα γραμμή και την τρέχουσα ανοχή, η συνάρτηση solveWithAdamsMoultonSolver εφαρμόζει την μέθοδο επίλυσης στην συγκεκριμένη γραμμή, Σχήμα 2.3.3.1.
- class_interpolation_method_1d.c: που επιστρέφει την γραμμική παρεμβολή σε ένα σημείο για κάποια δεδομένη χρονική στιγμή εφαρμόζοντας την συνάρτηση interpolate1DAt.
- class_solver_of_initial_value_problem_1d.c: συνάρτηση initialize_solver αρχικοποίησης δομής solverStruct. Περιλαμβάνει επίσης την συνάρτηση

updateHistoryArrayOfSolverOrder που διατηρεί την πληροφορία για την τάξη της μεθόδου στις προηγούμενες, από την τρέχουσα, χρονικές στιγμές

- class_specify_parameters.c: το αρχείο που περιέχει τις ρουτίνες που επιστρέφουν την πραγματική λύση, συνάρτηση f και derivative_f, τις τιμές των συναρτήσεων a και b (a_function, a_derivative, b_function, b_derivative) και την τιμή πεπερασμένης διαφοράς συνάρτηση g.



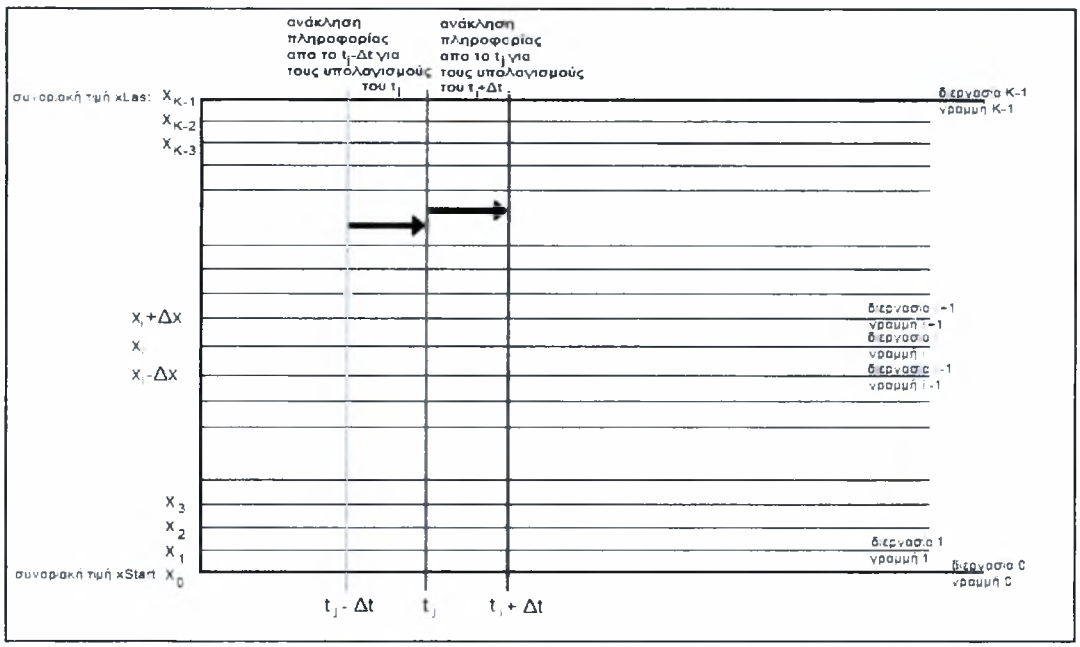
Σχήμα 2.3.3.1 Απεικόνιση των συνιστωσών της C-Mol εφαρμογής. Οι ακμές περιγράφουν τη ροή εκτέλεσης του προγράμματος.

Το μοντέλο της μεθόδου των γραμμών διέπεται από τους ακόλουθους περιορισμούς:

- Σε κάθε χρονική στιγμή να χρησιμοποιούμε το ίδιο Δx.
- Για κάθε τιμή του x να χρησιμοποιούμε το ίδιο Δt.
- Για κάθε γραμμή και κάθε χρονικό σημείο να χρησιμοποιούμε την ίδια μέθοδο επίλυσης συνήθης διαφορικής εξίσωσης, η οποία είναι η **Adams-Moulton**.
- Για να ακολουθήσουμε τα πλαίσια που ορίζει η απαιτούμενη ακρίβεια, προσαρμόζουμε τα Δx και Δt.

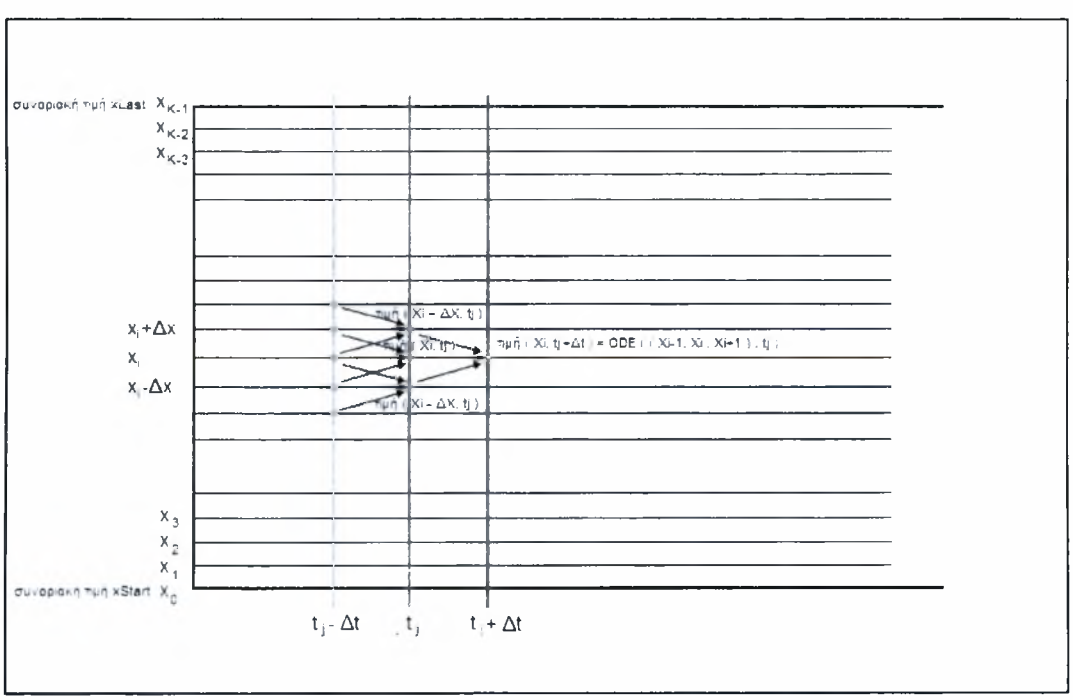
Ακολουθώντας τον τρίτο περιορισμό, η εφαρμογή σε κάθε γραμμή της μεθόδου επίλυσης συνήθης διαφορικής εξίσωσης Adams-Moulton, δημιουργεί ανάγκες για πληροφορία τιμών προηγούμενων υπολογιστικών βημάτων. Η τάξη της μεθόδου είναι αυτή που καθορίζει και την έκταση της πληροφορίας, αναφορικά με το πλήθος πόσων τιμών προηγούμενων υπολογιστικών βημάτων χρειάζεται η μέθοδος, για να πραγματοποιήσει τους υπολογισμούς του τρέχοντος υπολογιστικού βήματος.

Θεωρώντας ότι χρησιμοποιούμε την μέθοδο αυτή **στην πρώτη της τάξη**, για να ξεκινήσουν και να συνεχιστούν οι υπολογισμοί απαιτείται η **ανάκληση πληροφορίας** για το προηγούμενο υπολογιστικό βήμα. Εδώ πρέπει να υπενθυμίσουμε πως κατά την αρχικοποίηση του προβλήματος, οι τιμές στη χρονική στιγμή $t_0=0$ είναι γνωστές, καθώς η επίλυση μερικών διαφορικών εξισώσεων αποτελεί πρόβλημα αρχικών τιμών. Στο Σχήμα 2.3.3.2 απεικονίζεται η ανάγκη για ανάκτηση πληροφορίας από το προηγούμενο υπολογιστικό βήμα.



Σχήμα 2.3.2.2 Εφαρμογή μεθόδου Adams-Moulton 1ης τάξης . Για τον υπολογισμό των τιμών του t_j+Δt υπολογιστικού βήματος, απαιτούνται οι τιμές του t_j.

Αν συνυπολογίσουμε ότι στην διακριτοποιημένη διάσταση του χρόνου εφαρμόζουμε τον τύπο πεπερασμένων διαφορών που περιγράφονται από την σχέση (2.3.3) ή (2.3.4), η μορφή που παίρνει το πρόβλημα αποτυπώνεται στο Σχήμα 2.3.3.2, το οποίο σκιαγραφεί και την ανάκληση τιμών που εκτελεί ο επεξεργαστής για τους υπολογισμούς σε ένα υπολογιστικό βήμα . Την ανάκληση τιμών από τον επεξεργαστή για κάθε γραμμή, την αποκαλούμε **επικοινωνία μεταξύ των γραμμών**.



Σχήμα 2.3.3.3 Επικοινωνία μεταξύ γραμμών

Για να γίνεται εύκολα και χωρίς κόστος η επικοινωνία μεταξύ των γραμμών, χρησιμοποιούμε ένα μονοδιάστατο πίνακα στον οποίο αποθηκεύουμε τις τιμές των προηγούμενων υπολογιστικών βημάτων κάθε γραμμής. Έτσι για να υπολογίσουμε την τιμή της x_i γραμμής στο $t_j+\Delta t$ υπολογιστικό βήμα, ανατρέχουμε στον πίνακα αυτό και χρησιμοποιούμε τις τιμές του που αντιστοιχούν στις γραμμές $x_i-\Delta x$, x_i, $x_i+\Delta x$. Τις τιμές αυτές τις διατηρούμε στον πίνακα solutionTable[ ], που περιγράψαμε νωρίτερα σε αυτή την παράγραφο.

Τέλος, για να ικανοποιήσουμε τον τελευταίο περιορισμό, σχετικά με την προσαρμογή των Δx και Δt, όπως προστάζει η απαιτούμενη ακρίβεια, αναπτύσσουμε στο πρόγραμμα συναρτήσεις, στις οποίες χρησιμοποιούμε το εκτιμώμενο σφάλμα και αναλόγως αυξάνουμε ή μειώνουμε τα Δx και Δt. Με αυτές τις συναρτήσεις εκτελούμε τις απαιτούμενες προσαρμογές της μεθόδου των γραμμών, ώστε να ικανοποιήσουμε τις απαιτήσεις του προβλήματος και τις παρουσιάζουμε στην ακόλουθη υπό ενότητα.

2.4 Προσαρμοστικότητα

Για να περιγράψουμε την λειτουργία της συνάρτησης ελέγχου του Δt, θεωρούμε TN τη μέθοδο υπολογισμού της νέας τιμής u(x,t+Δt) από τα δεδομένα upast. Έτσι :

$$u(x,t+\Delta t) = TN(upast)$$

Ακόμα, συμβολίζουμε με utrue την πραγματική τιμή της u(x,t+Δt), που υπολογίζεται χρησιμοποιώντας τα δεδομένα upast και την πραγματική λύση που δίνεται από τη σχέση (2.3.2.1). Τότε το λάθος της συγκεκριμένης χρονικής στιγμής terr ([5]) εκτιμάται από την:

$$terr = | TN(upast) - TN(utrue) | \qquad (2.4.1)$$

Υποθέτοντας ότι η τιμή της τοπικής ανοχής ελέγχεται από την τιμή της :

$$loctol = Tol*(1+|u|) \qquad (2.4.2)$$

Όπου Tol είναι η καθορισμένη από το πρόβλημα, παράμετρος της ανοχής, και |u| το μέτρο της πραγματικής λύσης, που μελετώντας την (2.3.2.1) συμπεραίνουμε ότι μπορεί να είναι μικρότερη ή ίση με 3. Έτσι για να είναι αποδεκτό ένα υπολογιστικό βήμα πρέπει το σφάλμα του βήματος αυτού να είναι μικρότερο ή ίσο με την ανοχή της δεδομένης χρονικής στιγμής και άρα:

$$terr \leq loctol \text{ ή } terr \leq 4*Tol \qquad (2.3.2.5)$$

Πριν παρουσιάσουμε το πώς λειτουργεί η συνάρτηση ελέγχου του Δt πρέπει να σημειώσουμε πως από την αρχή του προβλήματος το Δt εμποδίζεται να γίνει πολύ μικρό, μικρότερο του 10^{-5} , καθώς μπορεί να οδηγήσει σε ατέρμονο βρόχο, με αποτέλεσμα το πρόβλημα να μην προχωρά. Ο έλεγχος του Δt γίνεται με:

Υπολογισμός terr, loctol

If terr>loctol and $\Delta t > 10^{-5}$ then $\Delta t = \Delta t/2$ and repeat time step

If terr<loctol/5 and Δt has been constant for the previous 4 time steps then $\Delta t = \Delta t * 2$

Αυτόν τον υπολογισμό τον εκτελούμε για κάθε γραμμή κάθε χρονική στιγμή. Από την στιγμή που τα βήματα είναι συγχρονισμένα, στο τελικό βήμα γίνεται :

$$\Delta t = \min (\Delta t \text{ for all lines })$$

Αντίστοιχα λειτουργεί και η συνάρτηση ελέγχου του Δx. Αν με XN δηλώσουμε την μέθοδο εκτίμησης του Δx για την διακριτοποίηση του προβλήματος, τότε το τοπικό χωρικό σφάλμα xerr είναι :

$$xerr = | XN(upast) - XN(utrue)|$$

Για την εκτίμηση του Δx ισχύει :

Υπολογισμός xerr, loctol

If xerr>loctol*4, then $\Delta x = \Delta x/2$ and reorganize lines

If xerr<loctol/10, then $\Delta x = \Delta x * 2$ and reorganize lines

Όταν αναφέρουμε ξανά-οργάνωση των γραμμών εννοούμε επανεκκίνηση των υπολογισμών με το νέο Δx. Αυτές τις συναρτήσεις τις συναντούμε στο αρχείο class_error_control_1d.c . Από την στιγμή που τα Δx και Δt αλλάζουν και ο επιλυτής χρησιμοποιεί τιμές για το $u(t_i,x)$ με t_i μικρότερο από τρέχον t, ορισμένες φορές το $u(t_i,x)$ μπορεί να μην υπολογιστεί. Σε αυτές τις περιπτώσεις εκτιμάμε το $u(t_i,x)$ με παρεμβολή. Για αυτό το λόγο έχουμε υλοποιήσει τη συνάρτηση getInnerSolutionAt στο αρχείο class_internal_solution_of_method_of_lines_1d.c.

Εδώ ολοκληρώνουμε την παρουσίαση του σειριακού αλγορίθμου της μεθόδου επίλυσης μερικών διαφορικών εξισώσεων των γραμμών. Η μέθοδος αυτή αποτελεί την κύρια δομή πάνω στην οποία στηρίξαμε την ανάπτυξη της παράλληλης υλοποίησης της μεθόδου των γραμμών, που είναι και το ζητούμενο της διπλωματικής. Στην επόμενη ενότητα περιγράφουμε αναλυτικά τη διαδικασία της ανάπτυξης της κατανεμημένης υλοποίησης της μεθόδου των γραμμών, τις τεχνολογίες που χρησιμοποιήσαμε και τα αποτελέσματα που είχε το εγχείρημά μας αυτό.

3

Παράλληλος Αλγόριθμος, Υλοποίηση και Αριθμητικά Αποτελέσματα

Τα τελευταία χρόνια έχει σημειωθεί τεράστια πρόοδος στην ανάπτυξη και την ανάλυση παράλληλων αλγορίθμων. Παράλληλοι αλγόριθμοι εφαρμόζονται σε προβλήματα που η σειριακή τους λύση είναι ήδη γνωστή και πολλές φορές έχουν σαν αποτέλεσμα την βελτίωση όχι μόνο της απόδοσης του προβλήματος, αλλά και του σειριακού αλγορίθμου στον οποίο βασίζονται.

Στο κεφάλαιο που ακολουθεί, κάνουμε την παρουσίαση και την ανάλυση της παράλληλης υλοποίησης της μεθόδου των γραμμών. Για την επίτευξη της κατανεμημένης αυτής εφαρμογής, χρησιμοποιήσαμε τη MOL – C εφαρμογή, που παρουσιάσαμε στην προηγούμενη ενότητα. Πιο συγκεκριμένα, ο κώδικας της C που αναπτύξαμε για την σειριακή εκτέλεση της μεθόδου των γραμμών, αποτελεί τον πυρήνα της κατανεμημένης υλοποίησης που δημιουργήσαμε στα πλαίσια της συγκεκριμένης διπλωματικής.

Αναλυτικότερα, στις υπό ενότητες που ακολουθούν, περιγράφουμε τις μεταβολές και επεκτάσεις που πράξαμε στην υλοποιημένη MOL – C σειριακή εφαρμογή, ώστε να την χρησιμοποιήσουμε ως μέθοδος επίλυσης μερικών διαφορικών εξισώσεων και σε κατανεμημένα υπολογιστικά περιβάλλοντα, με στόχο την βελτίωση της αποδόσεως του

αλγορίθμου. Επιπλέον, εμβαθύνουμε σε τμήματα κώδικα που εμφανίζουν αλγοριθμικό ενδιαφέρον και παρουσιάζουμε τα εργαλεία και τις τεχνολογίες που χρησιμοποιήσαμε για να επιτευχθεί αυτός ο σκοπός.

Για να γίνει πιο εύκολη η χρήση του λογισμικού που δημιουργήσαμε σε τρίτους, αναπτύσσεται μια ενότητα σε αυτό το κεφάλαιο, που αναλύουμε τις διαδικασίες εγκατάστασης και εκτέλεσης του κώδικα. Τέλος, αυτή η εργασία δεν θα είχε νόημα αν δεν παρουσιάζαμε αριθμητικές μετρήσεις, ώστε να μπορούμε να συγκρίνουμε την εφαρμογή με την αντίστοιχη σειριακή της εκδοχή.

3.1 Παράλληλα/Κατανεμημένα Υπολογιστικά Συστήματα

Οι αυξανόμενες απαιτήσεις υπολογιστικής δύναμης για τις ανάγκες ερευνητικών προγραμμάτων , υπολογισμών και εφαρμογών, έχουν οδηγήσει στην δημιουργία ενός νέου κλάδου της επιστήμης υπολογιστών, αυτή των παράλληλων υπολογισμών. Παράλληλος υπολογισμός είναι η επιστήμη που ασχολείται με την εκτέλεση πολλών εντολών ταυτόχρονα, λειτουργώντας σύμφωνα με την αρχή: «*μεγάλα προβλήματα μπορούν να χωριστούν σε μικρότερα, τα οποία επιλύονται ταυτόχρονα*». Οι παράλληλοι υπολογιστές που χρησιμοποιούνται για την επίτευξη αυτών των αναγκών διαχωρίζονται σύμφωνα με το επίπεδο του hardware στις ακόλουθες κατηγορίες ([11]) :

- **Πολυπύρηνους επεξεργαστές (multi-core processors)**: συνδυάζει 2 ή περισσότερους ανεξάρτητους πυρήνες σε έναν, συνθέτοντας ένα μόνο ολοκληρωμένο κύκλωμα επεξεργαστή.
- **Κατανεμημένο υπολογιστικό σύστημα (distributed computer)**: είναι ένα σύστημα υπολογιστών κατανεμημένης μνήμης, στο οποίο οι μονάδες του είναι διασκορπισμένες σε διάφορα γεωγραφικά σημεία, ενώ η επικοινωνία τους συμβαίνει μέσω ενός δικτύου ευρείας γεωγραφικής ζώνης. Οι υπολογιστικές του μονάδες δεν είναι απαραίτητο να εκτελούν τις ίδιες διεργασίες διαφορετικών δεδομένων, καθώς σε κάθε μία μπορεί να ανατεθεί διαφορετική λειτουργία μιας μεγάλης εργασίας.
- **Cluster**: είναι ένα σύνολο από πολλαπλές αυτόνομες μηχανές. Οι μηχανές που συνθέτουν ένα cluster είναι συνδεδεμένες μεταξύ τους μέσω γρήγορου τοπικού δικτύου και έτσι επικοινωνούν μεταξύ τους. Τα cluster συνήθως αναπτύσσονται για να βελτιώσουν την απόδοση και την διαθεσιμότητα που προσφέρει ένας μόνο υπολογιστής. Σε αυτές τις υπολογιστικές μονάδες, αντιστοιχίζεται η ίδια εργασία, με

διαφορετικά δεδομένα υπολογισμού. Επιλέγεται ως λύση καθώς είναι πιο οικονομικός από έναν υπολογιστή αντίστοιχων δυνατοτήτων, αν υπάρχει βέβαια.

- **Grid**: αποτελείται από υπολογιστές που συνεργάζονται μεταξύ τους για την επίλυση ενός προβλήματος το οποίο όμως συνθέτεται από πολλές ανεξάρτητες δουλειές ή πακέτα δουλειάς και έχουν ως μέσω επικοινωνίας το διαδίκτυο. Μια υπηρεσία του Grid αναλαμβάνει να κατανείμει και αναθέσει την δουλεία στους υπολογιστές, που θα την εκτελέσουν ανεξάρτητα από τις υπόλοιπες μονάδες του Grid. Σε αυτή τη μορφή υπολογιστικού συστήματος τα ενδιάμεσα αποτελέσματα της εργασίας που έχει ανατεθεί σε έναν υπολογιστή δεν επηρεάζουν την δουλεία άλλων κόμβων του Grid.

Σε αυτά τα υπολογιστικά περιβάλλοντα που αναφέραμε προηγουμένως, ένας αλγόριθμος για να επωφεληθεί από τις δυνατότητες που παρέχουν, πρέπει να είναι κατάλληλων προδιαγραφών. Οι αλγόριθμοι που εκτελούνται στα κατανεμημένα υπολογιστικά συστήματα, καλούνται **παράλληλοι αλγόριθμοι**.

Ένας παράλληλος αλγόριθμος, αντίθετα με τους σειριακούς, μπορεί να εκτελεστεί σε κομμάτια, ταυτόχρονα σε διαφορετικές συσκευές επεξεργασίας και έπειτα να συλλεχθούν τα επιμέρους αποτελέσματα, σχηματίζοντας το επιθυμητό. Οι παράλληλοι αλγόριθμοι είναι προτιμητέοι, καθώς αν είναι ορθά δομημένοι, επιταχύνουν την εκτέλεση μεγάλων υπολογιστικών ζητημάτων, συγκριτικά με τους ακολουθιακούς, εξαιτίας του ταυτόχρονου τρόπου εκτέλεσης, παρέχοντας μεγαλύτερη απόδοση.

Παρά τα πλεονεκτήματα των παράλληλων εφαρμογών, υπάρχουν και περιορισμοί. Η παράλληλη μετατροπή ενός αλγορίθμου προσφέρει ταχύτητα εκτέλεσης, όμως ο σχεδιασμός και η υλοποίηση του δεν είναι εύκολη υπόθεση. Αν δεν γίνει σωστά, μπορεί να οδηγήσει σε εφαρμογές με λανθασμένα αποτελέσματα και μικρότερη απόδοση από αυτή των σειριακών. Ακόμα, κάθε παράλληλος αλγόριθμος έχει ένα οριακό σημείο αποδόσεως, το οποίο όταν υπερβεί, η κλιμάκωση του σε μεγαλύτερο επίπεδο παραλληλίας δεν επιφέρει σημαντική χρονοβελτίωση (**σημείο κορεσμού - saturation – Amdahl's point**), συγκριτικά με το κόστος.

Το κόστος ή η πολυπλοκότητα των σειριακών αλγορίθμων εκτιμάται σε δέσμευση μνήμης και σε χρόνο, αναφορικά με τους κύκλους μηχανής. Στους παράλληλους κώδικες πρέπει να

συνυπολογιστεί και το κόστος επικοινωνίας μεταξύ των επεξεργαστών, που εκτελούν αυτή την εφαρμογή.

Συνεπώς, οι παράλληλοι αλγόριθμοι πρέπει να ακολουθούν κάποιες βασικές αρχές σχεδιασμού, για να μπορέσουν να επιλύσουν αποδοτικά τα προβλήματα παράλληλα. Αυτές οι βασικές αρχές είναι ([10]) :

- **Τεμαχισμός (partition- decomposition)**: Ο διαμερισμός του προβλήματος σε μικρότερες ανεξάρτητες εργασίες που μπορούν και εκτελούνται ταυτόχρονα.
- **Επικοινωνία (communication)** : ορισμός επικοινωνίας μεταξύ των επιμέρους εργασιών.
- **Συγχώνευση (agglomeration)** : συγχώνευση μικρών επιμέρους εργασιών σε μία εργασία με σκοπό την μείωση του επικοινωνιακού κόστους
- **Αντιστοίχηση (mapping)** : αντιστοίχιση εργασιών σε επεξεργαστές με έμφαση στην ταυτόχρονη εργασία και στο ελάχιστο επικοινωνιακό κόστος.

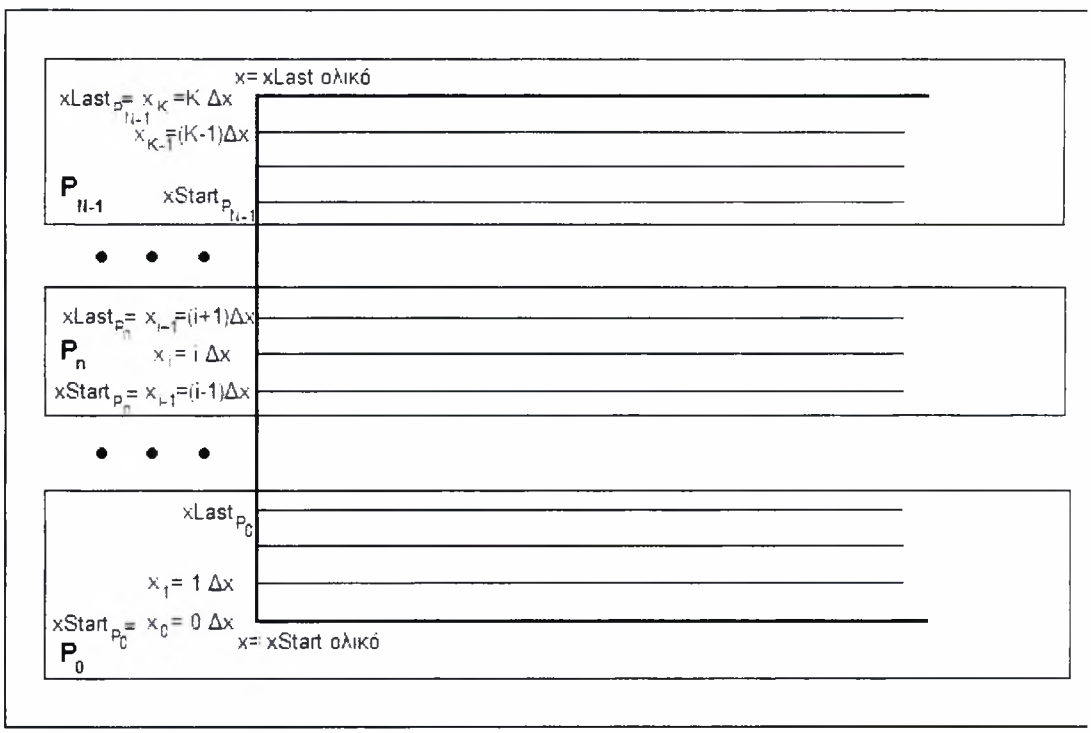
Στην επόμενη ενότητα αναλύουμε τον παράλληλο αλγόριθμο που αναπτύχθηκε σε αυτή την διπλωματική εργασία και παρουσιάζουμε τη συμφωνία του με τις απαιτούμενες προδιαγραφές σχεδιασμού παράλληλου αλγορίθμου.

3.2 Παράλληλος Αλγόριθμος

Όπως περιγράψαμε στο προηγούμενο κεφάλαιο στην εν λόγω διπλωματική εργασία επιλύουμε μερικές διαφορικές εξισώσεις του τύπου (2.3.1) σε παράλληλο υπολογιστικό σύστημα όπως είναι το cluster, Centaurus του τμήματος. Την περιγραφή του cluster την κάνουμε σε ακόλουθη ενότητα, στην οποία αναλύουμε της πλατφόρμες και τα εργαλεία που χρησιμοποιήσαμε. Για να ακολουθήσουμε τις βασικές αρχές παράλληλων αλγορίθμων, που μόλις αναφέραμε και να τις εφαρμόσουμε στον ήδη υλοποιημένο σειριακό αλγόριθμο της C-Mol, εκτελούμε τα παρακάτω βήματα:

1. **ΤΕΜΑΧΙΣΜΟΣ :** Πρέπει αρχικά να τεμαχίσουμε το πεδίο στο οποίο θα εφαρμοστεί η παράλληλη υλοποίηση. Όπως αναφέραμε στην προηγούμενη ενότητα, το πεδίο στο οποίο μας απασχολεί η επίλυση της μερικής διαφορικής εξίσωσης αναπαρίσταται από το Σχήμα 2.3.1. Αυτό το πεδίο θέλουμε να το τεμαχίσουμε και να το αναθέσουμε στους διαθέσιμους επεξεργαστές, που θα εκτελέσουν την παράλληλη εφαρμογή της

MOL – C. Συνεπώς, το πεδίο διαιρείται σε τόσο μέρη, όσο είναι το πλήθος των διαθέσιμων επεξεργαστών. Έστω N το πλήθος αυτό, έτσι το σχήμα 2.3.1 λαμβάνει την μορφή του Σχήματος 3.2.1.



Σχήμα 3.2.1 Τεμαχισμός προβλήματος

Για να γίνει κατάλληλα ο τεμαχισμός του προβλήματος, πρέπει το πλήθος των γραμμών που σχηματίζονται μετά από την διαδικασία της διακριτοποίησης που εφαρμόζουμε στον αλγόριθμο της μεθόδου των γραμμών (βλέπε 2.3.1), να το μοιράσουμε όσο το δυνατόν πιο ισότιμα στους επεξεργαστές. Έστω K (παράμετρος εισόδου) το πλήθος των διαστημάτων στο οποίο διαιρούμαι το πεδίο $xStart_{ολικό}$ – $xLast_{ολικό}$. Τότε, K+1 θα είναι το πλήθος των γραμμών, από 0-K , που σχηματίζονται από το σημείο $xStart_{ολικό}$ μέχρι το $xLast_{ολικό}$. Ο αριθμός των γραμμών που πρέπει να αναθέσουμε σε κάθε επεξεργαστής είναι:

$$\text{Lines per } P_i = \frac{K+1}{N} = L \qquad (3.2.1)$$

Σε αυτό το σημείο προκύπτουν δύο σημαντικά ζητήματα:

a. **L είναι δεκαδικός**: Τροποποιούμε την σχέση (3.2.1) ως ακολούθως:

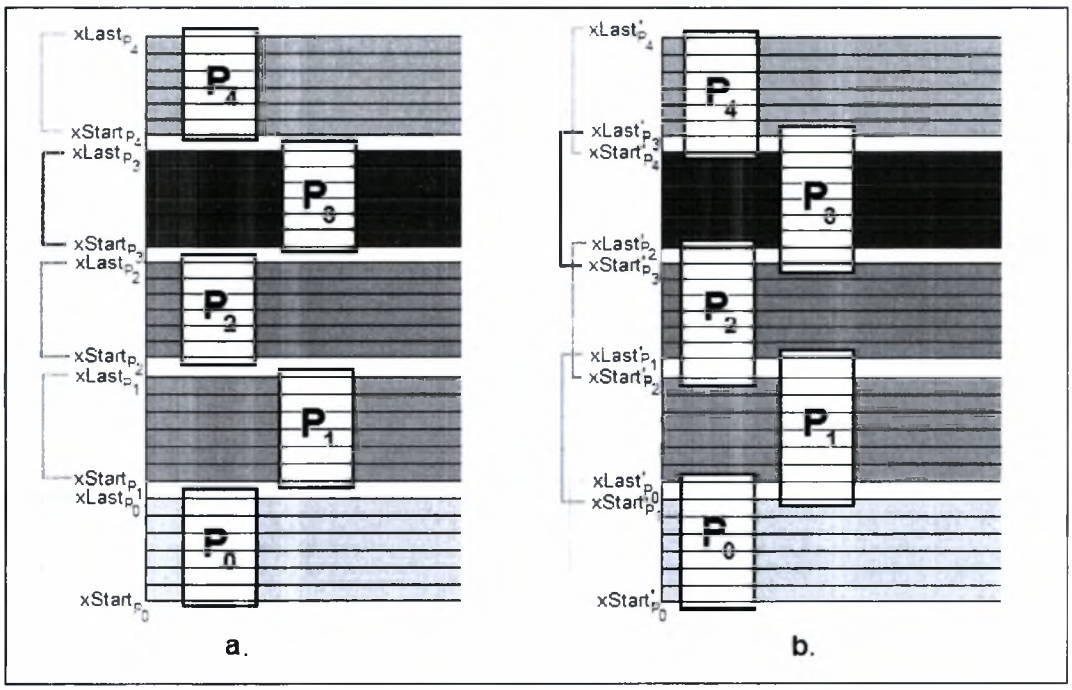
$$\text{Lines per } P_i = \left\lceil \frac{K+1}{N} \right\rceil = L \qquad (3.2.2)$$

Σε αυτή την περίπτωση, κάθε επεξεργαστής θα πάρει από μία γραμμή παραπάνω. Αυτό μπορεί να έχει ως αποτέλεσμα ο τελευταίος επεξεργαστής να λάβει γραμμές έξω από το όριο που ορίζει το πρόβλημα, $xLast_{ολικό}$. Για να αποφύγουμε τέτοιες καταστάσεις επιλέγουμε το πλήθος των γραμμών που λαμβάνει ο τελευταίος επεξεργαστής είναι:

$$P_{N-1} = (K+1) - [(N-2) * L] \qquad (3.2.3)$$

b. **Εφαρμογή MOL-C και συνοριακές συνθήκες – γραμμές**. Όταν έχουμε πλήθος γραμμών K+1 στη σειριακή εφαρμογή που παρουσιάσαμε παραπάνω, η μέθοδος επίλυσης συνήθων διαφορικών εξισώσεων Adams-Moulton εφαρμόζεται για K-1 εσωτερικές γραμμές, καθώς οι δύο εξωτερικές, $xStart_{ολικό}$ και $xLast_{ολικό}$,αποτελούν τις συνοριακές συνθήκες.

Εκτελώντας κάθε επεξεργαστής P_i, όπου i=0-(N-1), τον αλγόριθμο MOL-C, στο πεδίο που του έχουμε αναθέσει, θα κάνει χρήση του ODE επιλυτή σε L-2 γραμμές και όχι σε L. Αυτό συμβαίνει γιατί τις γραμμές $xStart_{Pi}$ και $xLast_{Pi}$, ο αλγόριθμος τις θεωρεί συνοριακές, γεγονός που οδηγεί σε λανθασμένη επίλυση της μερικής διαφορικής εξίσωσης, καθώς οι γραμμές αυτές για το αρχικό πρόβλημα, είναι εσωτερικές. Το Σχήμα 3.2.2.a περιγράφει αυτό το ζήτημα.



Σχήμα 3.2.2. a. Πρόβλημα που προκύπτει με τις οριακές γραμμές κάθε επεξεργαστή με την εφαρμογή της σειριακής εφαρμογής της Mol-C κατά τον τεμαχισμό του προβλήματος. b. Μέθοδος που εφαρμόζεται στον παράλληλο αλγόριθμο για την αντιμετώπιση του προβλήματος τεμαχισμού.

Για να αντιμετωπίσουμε αυτό το ζήτημα, παριστάνουμε τις νέες συνοριακές συνθήκες με $xStart''_{Pi}$ και $xLast''_{Pi}$, για να τις ξεχωρίσουμε από τις προηγούμενες $xStart_{Pi}$, $xLast_{Pi}$ και εφαρμόζουμε την ακόλουθη μεθοδολογία:

- Επεξεργαστής 0 : $xStart''_{P0} = xStart_{ολικο}$, $xLast''_{P0} = xStart_{P1}$
- Ενδιάμεσοι επεξεργαστές, 1-(N-2):

$$xStart''_{Pi} = xLast_{Pi-1} \text{ και } xLast''_{Pi} = xStart_{Pi+1}$$

- Επεξεργαστής N : $xStart''_{P_N} = xLast_{P_{N-1}}$ και $xLast_{P_N} = xLast_{ολικο}$

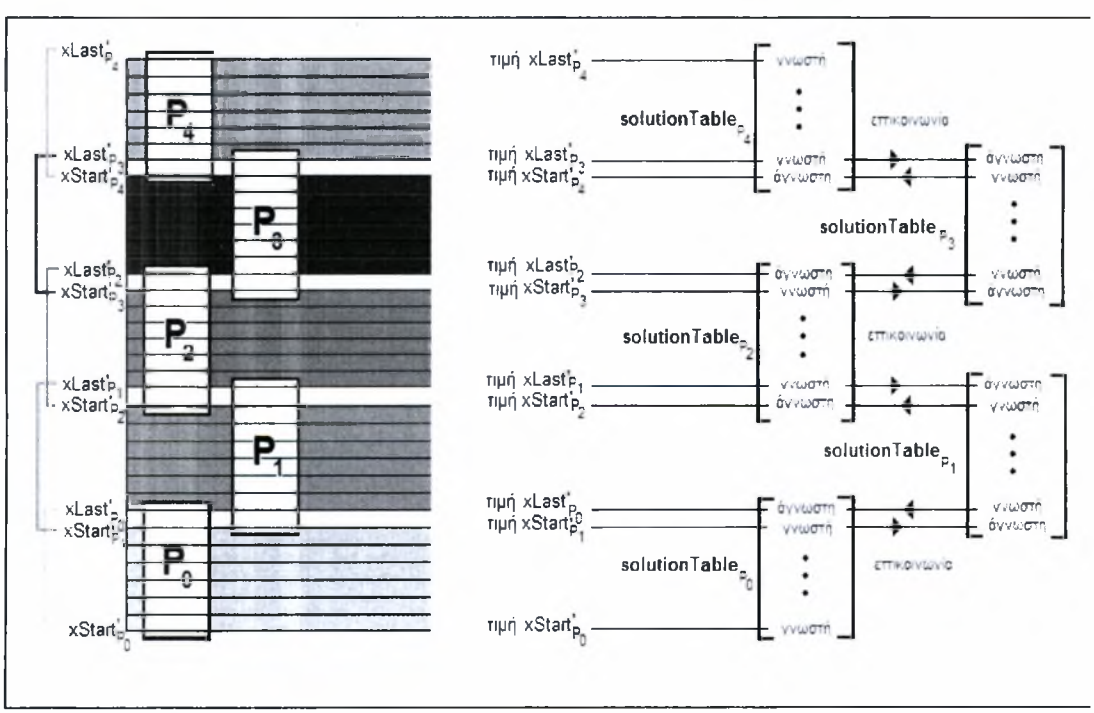
Η γραφική απεικόνιση της μεθόδου αυτής που εφαρμόζουμε στον παράλληλο αλγόριθμο βρίσκεται στο Σχήμα 3.2.2.b.

Συνεπώς, κάθε επεξεργαστής αναλαμβάνει L+2 γραμμές και εκτελεί την μεθοδολογία της σειριακής Mol-C για τις L εσωτερικές του γραμμές. Τις τιμές στις συνοριακές γραμμές ο Mol-C αλγόριθμος τις θεωρεί γνωστές. Ο

κάθε επεξεργαστής P_i, δεν τις γνωρίζει, όμως τις έχουν υπολογίσει οι γειτονικοί του επεξεργαστές καθώς για αυτούς αποτελούν εσωτερικές. Για να τις αποκτήσει ο P_i. τις ζητάει από αυτούς. Έτσι περνάμε στο ζήτημα της **επικοινωνίας** της παράλληλης υλοποίησης της μεθόδου των γραμμών.

2. **ΕΠΙΚΟΙΝΩΝΙΑ**: Στον σειριακό αλγόριθμο της C-Mol, η επικοινωνία μεταξύ των διεργασιών, ώστε να εκτελέσει ο επεξεργαστής τους υπολογισμούς του συστήματος (2.3.2.3) με την μέθοδο επίλυσης ODE, Adams-Moulton, γίνεται με την χρήση του πίνακα solutionTable που διατηρεί ο επεξεργαστής για τις υπολογισμένες τιμές των προηγούμενων υπολογιστικών βημάτων. Έναν τέτοιο πίνακα $\text{solutionTable}_{P_i}$ διατηρεί και κάθε επεξεργαστής P_i, για τις τιμές των εσωτερικών γραμμών.

Ένα στιγμιότυπο αυτού του πίνακα στην χρονική στιγμή t_i, πριν την εκτέλεση των υπολογισμών του επόμενου υπολογιστικού βήματος t_i,$+\Delta t$, για τους επεξεργαστές P_i στις γραμμές $xStart''_{P_i}$ ως $xLast''_{P_i}$ που αναλογούν στον καθένα, παρουσιάζεται στο Σχήμα 3.2.3.



Σχήμα 3.2.3. Επικοινωνία επεξεργαστών για ανταλλαγή συνοριακών γειτονικών τιμών, που βρίσκονται αποθηκευμένες στους πίνακες solutionTable του κάθε επεξεργαστή

3. **ΣΥΓΧΩΝΕΥΣΗ-ΑΝΤΙΣΤΟΙΧΙΣΗ**: Οι δύο αυτές ιδιότητες που πρέπει να ικανοποιεί ο παράλληλος αλγόριθμος, γίνονται με έμφαση το ελάχιστο επικοινωνιακό κόστος και την ταυτόχρονη εργασία. Στην ταυτόχρονη υλοποίηση της μεθόδου των γραμμών, όπως έχουμε αναφέρει, το πλήθος των διεργασιών που δημιουργεί το καθορίζει το σύνολο των διαθέσιμων επεξεργαστών N.

Ωστόσο, θα μπορούσαμε το αρχικό πρόβλημα να το διαμερίσουμε σε μεγαλύτερου πλήθους ισοδύναμες διεργασίες από όσους πόρους επεξεργασίας διαθέτουμε. Έστω M το πλήθος των διεργασιών, Y ο χρόνος εκτέλεσης ενός υπολογιστικού βήματος μίας διεργασίας από έναν επεξεργαστή (θεωρούμε ότι οι επεξεργαστές έχουν τις ίδιες υπολογιστικές δυνατότητες), $T_{M=N}$ ο χρόνος επικοινωνίας των επεξεργαστών όταν οι διεργασίες είναι αριθμητικά ίσες με το πλήθος των επεξεργαστών, και T_M όταν είναι διαφορετικό. Τότε για $M > N$:

- Αν $M \neq q*N$, με q ακέραιος διαφορετικός της μονάδας, σε ορισμένους από τους επεξεργαστές P_r θα αναθέταμε περισσότερες διεργασίες, από τους υπολοίπους $(N-P_r)$. Αυτό θα είχε αντίκτυπο :
    1. **Στην ταυτόχρονη εργασία**. Οι επεξεργαστές με λιγότερες διεργασίες να περιμένουν αυτούς με τις περισσότερες, για να προχωρήσουν στο επόμενο υπολογιστικό βήμα. Αυτό συμβαίνει διότι ο χρόνος εκτέλεσης από τους P_r θα ήταν μεγαλύτερος από τον αντίστοιχο των $(N-P_r)$.
    2. **Στην επικοινωνία των επεξεργαστών.** Αυτό είναι εύκολα αντιληπτό καθώς , περισσότερες διεργασίες σημαίνει αυτόματα και περισσότερη επικοινωνία μεταξύ γειτονικών επεξεργαστών για ανταλλαγή συνοριακών τιμών.
- Αν $M=q*N$, με q ακέραιος διαφορετικός της μονάδας, τότε η ταυτόχρονη εργασία θα έμενε ανεπηρέαστη αλλά ο χρόνος επικοινωνίας πάλι θα αυξανόταν.

Αντίθετα, αν το πλήθος των διεργασιών ήταν μικρότερο από το πλήθος των επεξεργαστών, όπως θα επίτασσε μια ενδεχόμενη συγχώνευση, τότε ορισμένοι από τους διαθέσιμους επεξεργαστές θα έμεναν χωρίς εργασία, με αντίκτυπο την ελάττωση της παραλληλίας. Ακολουθώντας αυτή τη συλλογιστική πορεία, φτάνουμε στο συμπέρασμα ότι ο τρόπος τεμαχισμού που επιλέξαμε για την υλοποίηση του παράλληλου

αλγορίθμου είναι αυτός που προστάζουν οι αρχές σχηματισμού ταυτόχρονου προγράμματος.

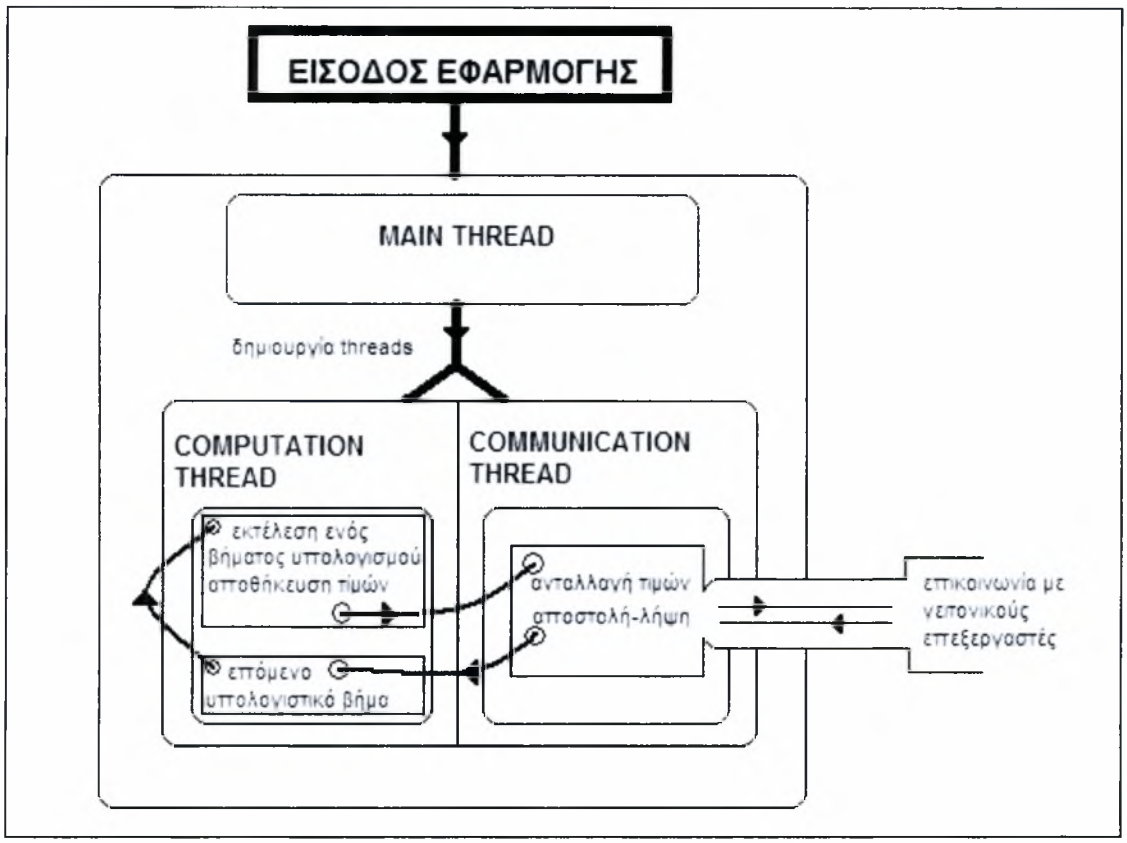
Αφού κατορθώσαμε να δώσουμε στον αλγόριθμό μας την μορφή παράλληλης εφαρμογής, περνάμε στα ζητήματα της υλοποίησης του που έχουν να κάνουν με τις λειτουργίες που πράττει κάθε επεξεργαστής και είναι οι εξής:

- **Λειτουργία Υπολογισμών**: η οποία γίνεται με εφαρμογή της μεθόδου των γραμμών στο πεδίο, που του έχει ανατεθεί. Για να εκτελέσει αυτή τη λειτουργία ο επεξεργαστής, επιλέξαμε να εφαρμόζει την σειριακή υλοποίηση της μεθόδου των γραμμών. Έτσι, κατά αντιστοιχία με την Mol – C που εφαρμόζει την μέθοδο των γραμμών σε όλα τα διακριτοποιημένα σημεία, ο επεξεργαστής P_i την εφαρμόζει για στα L σημεία που του αντιστοιχούν.
- **Λειτουργία Επικοινωνίας Επεξεργαστών**: όπου κάθε επεξεργαστής ανταλλάσει με τους γειτονικούς του τις συνοριακές τιμές, μετά από κάθε υπολογιστικό βήμα.

Για να επιτευχθεί αυτός ο σκοπός στον παράλληλο κώδικα της εφαρμογής δημιουργήσαμε ένα **thread** εκτέλεσης της κάθε μιας λειτουργίας ξεχωριστά:

- Το **communication thread**, που αναλαμβάνει την ανταλλαγή τιμών μεταξύ γειτονικών επεξεργαστών
- Το **computation thread**, που εκτελεί την Mol-C εφαρμογή

Αυτά τα δύο νήματα γεννιούνται από το **main thread**, που ξεκινά να τρέχει στον κάθε επεξεργαστή, όταν αυτός εκτελεί τον παράλληλο κώδικα που κατασκευάσαμε. Στο Σχήμα 3.2.4 απεικονίζουμε τα threads που δημιουργεί ένας επεξεργαστής και τη ροή εκτέλεσής τους, όπως την έχουμε αναπτύξει. Επίσης, από το Σχήμα 3.2.4 αποτυπώνουμε την ανάγκη εναλλαγής μεταξύ των επιμέρους thread. Αυτό το ζήτημα θα αναλυθεί στην ενότητα που ακολουθεί και παρουσιάζει λεπτομερώς ζητήματα υλοποίησης.



Σχήμα 3.2.4 Ροή εκτέλεσης και κατάσταση threads σε κάθε επεξεργαστή.

3.3 Υλοποίηση

3.3.1 Λεπτομέρειες υλοποίησης

Σε αυτή την ενότητα περιγράφουμε λεπτομερώς θέματα υλοποίησης της διπλωματικής εργασίας, που έχουν αλγοριθμικό ενδιαφέρον. Για κάθε ένα από τα ζητήματα αυτά έχουμε αναπτύξει μια ενότητα στην οποία το περιγράφουμε αναλυτικά. Στην ανάπτυξη αυτή ακολουθήσαμε τη μέθοδο από έξω προς τα μέσα, όσον αναφορά την μελέτη των ζητημάτων, δηλαδή περιγράφουμε τα ζητήματα με τη σειρά που συναντούνται στην εφαρμογή μας, σύμφωνα με τη ροή εκτέλεσης.

3.3.1.1 center_node.py

Σε κάθε επεξεργαστή, ο οποίος εκτελεί την παράλληλη μορφή της μεθόδου των γραμμών, πρέπει να προσδιορίζουμε το πεδίο στο οποίο θα εκτελέσει τους υπολογισμούς της μερικής διαφορικής εξίσωσης. Αυτή τη δουλεία, στη σειριακή εφαρμογή την εκτελούμε, όπως αναφέραμε στην προηγούμενη ενότητα (Κεφ. 2.3.2), με το αρχείο params.txt. Συνεπώς, από τη στιγμή που το νήμα των υπολογισμών κάθε επεξεργαστή, εκτελεί την σειριακή εφαρμογή

όπως αυτή μετατράπηκε για τις ανάγκες της κατανεμημένης, πρέπει να περνιέται ως παράμετρος ένα αρχείο (processorN.txt) παρόμοιο με το params.txt που να προσφέρει τα απαραίτητα δεδομένα στο πρόγραμμα.

Για την δημιουργία αυτών των αρχείων εισόδου κάθε επεξεργαστή, αναπτύσσουμε το αρχείο center_node.py. Με την εκτέλεση αυτού πραγματοποιούμε τις ακόλουθες λειτουργίες:

1. Εισάγουμε το αρχείο που θέλουμε να ανοίξουμε και να διαβάσουμε. Στην περίπτωσή μας. το params.txt.
2. Έπειτα εισάγουμε το πλήθος των αρχείων με όνομα processorsi.txt, όπου i=0-N-1 (N το πλήθος των επεξεργαστών) που προ-υπάρχουν στο φάκελο εκτέλεσης. Δίνοντας τον αριθμό αυτό, διαγράφουμε αυτόματα όλα τα ήδη υπάρχοντα αρχεία με αυτή την ονομασία. Αυτή τη λειτουργία την κάνουμε, ώστε να μην υπάρξει ενδεχόμενο, κάποιος επεξεργαστής να πάρει ως είσοδο του προβλήματος ένα αρχείο που δεν του αντιστοιχεί.
3. Η τρίτη εισαγωγή που μας ζητείται, έχει να κάνει με το πλήθος των επεξεργαστών στο οποίο θα διαμεριστεί το πρόβλημα, δηλαδή το πλήθος των αρχείων που πρέπει να δημιουργήσουμε.
4. Έπειτα και από την εισαγωγή του προηγούμενου βήματος του αρχείου αυτού, έχουμε κατασκευάσει αυτόματα τα αρχεία εισόδου κάθε επεξεργαστή, που είναι διαθέσιμος για να εκτελέσει την παράλληλη εφαρμογή. Για την κατασκευή αυτή γίνεται κλήση της συναρτήσεως createFiles που αποτελεί μέρος του αρχείου processorFile.

Μετά από αυτή την διαδικασία, στον γονικό φάκελο που βρισκόμαστε, εμφανίζονται τα αρχεία εισόδου των επεξεργαστών, processorsi.txt, που προσανατολιζόμαστε να χρησιμοποιήσουμε στη παράλληλη εφαρμογή. Πλέον μπορούμε να εκτελέσουμε τον κατανεμημένο κώδικα που δημιουργήθηκε για τις ανάγκες αυτής της διπλωματικής.

3.3.1.2 each_node.py

Η παράλληλη εφαρμογή ξεκινά με την εκτέλεση του αρχείου each_node.py. Για να την εκκινήσουμε, πληκτρολογούμε την εντολή:

```
>>>lamboot nodes                    σηκώνει τους κόμβους του cluster
>>>mpirun –np num_of_processors mpipython each_node.py
```

Με το που δίνουμε την τελευταία εντολή στο κατανεμημένο υπολογιστικό σύστημα στο οποίο προσανατολιζόμαστε να εκτελέσουμε την μέθοδο των γραμμών, σε κάθε επεξεργαστή

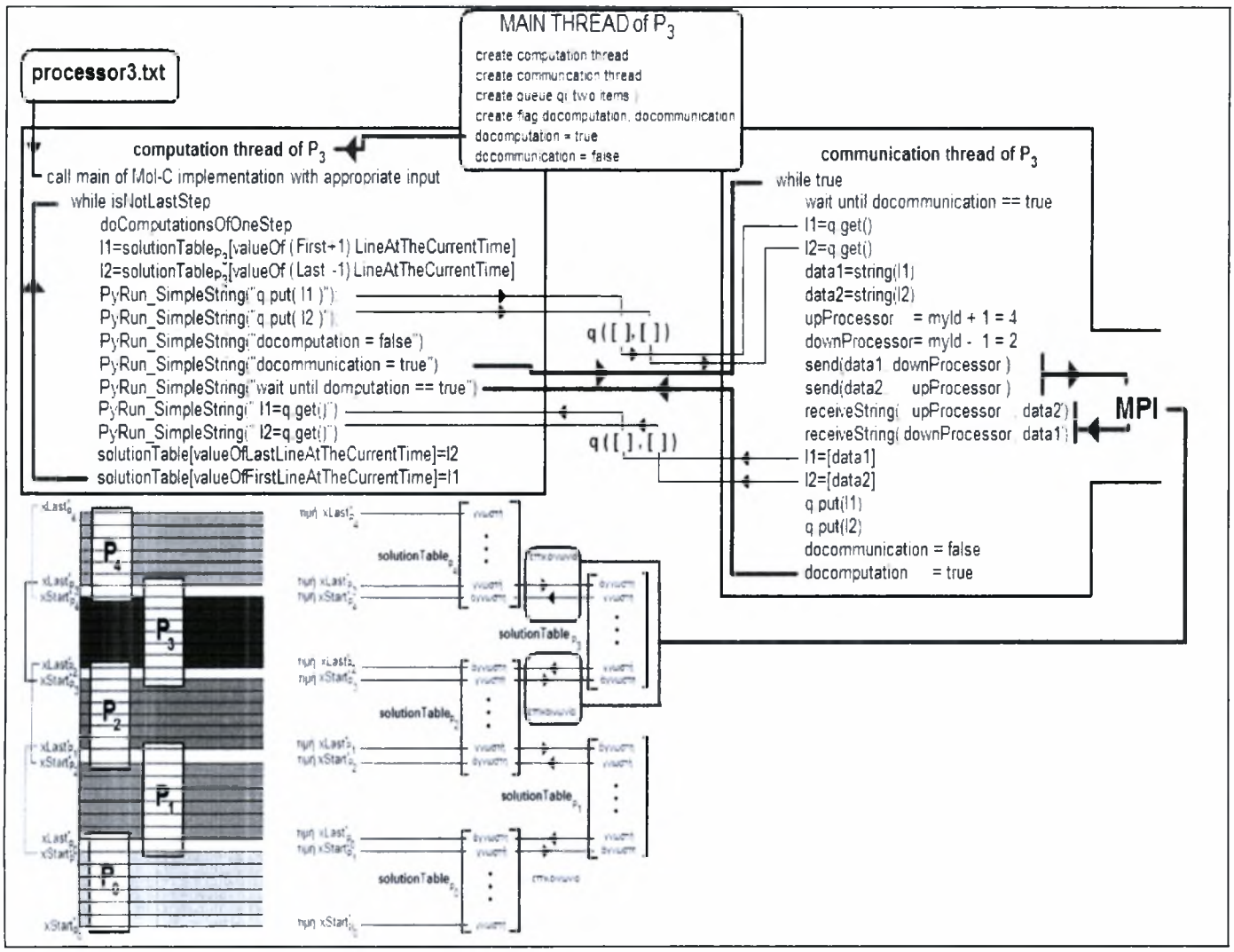
γεννάμε το main thread, μέσω του οποίου φέρνουμε εις πέρας, τα ζητήματα του παράλληλου αλγορίθμου. Όπως περιγράψαμε στο Σχήμα 3.2.4, από το main thread δημιουργούμε δύο threads, στα οποία αναθέτουμε τις καίριες λειτουργίες του παράλληλου αλγορίθμου.

Στη συνέχεια αναλαμβάνουμε το ξύπνημά τους με κατάλληλη σειρά, ώστε πρώτα να γίνει η αρχικοποίηση της σειριακής μεθόδου των γραμμών, η εκτέλεση του πρώτου βήματος υπολογισμών από το computation thread και έπειτα η επικοινωνία, με το communication thread. Στο main thread προετοιμάζουμε τις πληροφορίες που πρέπει να γνωρίζουμε για τα δύο νήματα, ώστε να είναι εφικτή η εκτέλεση τους. Με τον όρο πληροφορίες, αναφερόμαστε στον κόσμο που πλαισιώνει την κατανεμημένη εφαρμογή και τα μέσα που διαθέτει ο κόσμος, ώστε τα δύο υποκείμενα στο main, threads, να εκτελέσουν τις λειτουργίες τους. Ουσιαστικά αναφερόμαστε στην ικανότητα:

- Να αντιληφθεί ο κάθε επεξεργαστής τον κόσμο γύρο του, δηλαδή τους επεξεργαστές με τους οποίους θα συνεργαστεί για να εκτελέσει την παράλληλη εφαρμογή (MPI.world)
- Να αποκτήσει ένα αποκλειστικό αναγνωριστικό το οποίο θα τον κάνει διακριτό και αντιληπτό από τους υπόλοιπους επεξεργαστές που πλαισιώνουν τον κόσμο.(id)
- Να κατασκευάσει μια δομή, αναγνωρίσιμη και από τα δύο threads που θα δρα ως μέσω επικοινωνίας αυτών των δύο. Αυτή η δομή είναι μία ουρά (FIFO- first in first out) με την ονομασία **q**, στην οποία τοποθετούμε τα δεδομένα(συνοριακές τιμές) που πρέπει το computation thread να μεταδώσει στο communication thread και αυτό με τη σειρά του να αποστείλει στους κατάλληλους γείτονες που τα χρειάζονται. Επιπλέον, στην q τοποθετούμε τις πληροφορίες που λαμβάνει το communication thread από τους γειτονικούς επεξεργαστές και πρέπει να τις μεταδώσει με την σειρά του στο computation thread, ούτως ώστε να μπορέσει να συνεχίσει τους υπολογισμούς του με τα κατάλληλα δεδομένα.
- Να δημιουργήσει ελεγκτές, ώστε να εναλλάσσουμε τα δύο threads, για την εκπλήρωση των λειτουργιών τους. Αυτοί οι ελεγκτές ονομάζονται docommunication και docomputation. Όταν ένας από τους δύο ελεγκτές έχει την τιμή true τότε το αντίστοιχο thread δεσμεύει τον επεξεργαστή και εκτελεί ένα βήμα της λειτουργίας του.

Την αναπαράσταση των τριών threads, μαζί με κομμάτια ψευδοκώδικα, που περιγράφουν τις λειτουργίες τους και τη ροή που ακολουθεί το πρόγραμμα (κόκκινη γραμμή) την παρουσιάζουμε στο Σχήμα 3.3.1. Όπως γίνεται αντιληπτό από την παρατήρηση του Σχήματος

το main thread επιτρέπει την δέσμευση του επεξεργαστή από το computation thread, που παρουσιάζεται στο ακόλουθο κεφάλαιο.



Σχήμα 3.3.1 Απεικόνιση της ροής εκτέλεσης του προγράμματος στον P3 επεξεργαστή, μαζί με τον ψευδοκώδικα που περιγράφει τις λειτουργίες του κάθε thread.

3.3.1.3 Συνάρτηση computation του each_node.py.

Στο αρχείο each_node.py, έχουμε υλοποιήσει τη συνάρτηση computation που αποτελεί τη λειτουργία του computation thread. Στη συνάρτηση αυτή κάνουμε κλήση του module _interface.so της σειριακής εφαρμογής της Mol-C, που έχουμε δημιουργήσει μέσω μιας αυτοματοποιημένης διαδικασίας με τη χρήση του εργαλείου SWIG. Τόσο τη διαδικασία αυτή, όσο και το εργαλείο SWIG τα παρουσιάζουμε σε επόμενες υπό-ενότητες. Εδώ, μας απασχολεί μόνο το γεγονός ότι στο computation thread κάνουμε χρήση της Mol-C εφαρμογής, εκτελώντας την από το αρχικό της βήμα, μέχρι το τελευταίο.

Για να γίνει αυτό, σε κάθε επεξεργαστή καλούμε την συνάρτηση main του _interface.so με το αρχείο εισόδου που του αντιστοιχεί (π.χ. Σχήμα 3.3.1 processor3.txt). Έτσι ξεκινάμε σε κάθε επεξεργαστή την εκτέλεση της Mol-C εφαρμογής, που περιγράψαμε στο προηγούμενο κεφάλαιο, με ορισμένες μεταβολές. Μετά από κάθε βήμα υπολογισμού, από τον εκάστοτε επεξεργαστή (P_3) πρέπει να μεταδώσουμε τις επόμενες, των συνοριακών του, τιμές, στους γειτονικούς του επεξεργαστές (P_2 και P_4) και να λάβουμε από αυτούς τις συνοριακές τους τιμές που δεν γνωρίζουμε μέχρι αυτό το σημείο. Για αυτές τις λειτουργίες εκτελούμε τα ακόλουθα βήματα:

1. αποθηκεύουμε τις τιμές των γραμμών (First+1) και (Last-1), τη δεδομένη χρονική στιγμή σε δύο μεταβλητές της C, l1 και l2, που είναι ορατές και από την Python, μέσω των εντολών :
    - l1=solutionTable[valueOf(First+1)LineAtTheCurrentTime]
    - l2= solutionTable[valueOf(Last-1)LineAtTheCurrentTime]
2. εισάγουμε τις δύο μεταβλητές αυτές στην ουρά q που δημιουργήσαμε στο main thread. Αυτό το κάνουμε από το περιβάλλον της Python. Για να γίνει αντιληπτό από τον σειριακό κώδικα της Mol-C, που μέχρι τώρα εκτελούμε στο computation thread, περνάμε την εντολή q.put(l1) και q.put(l2), ως παράμετρο της Python/C API συνάρτησης, PyRun_SimpleString(""). Αυτή την Python/C API τη χρησιμοποιούμε για κάθε δήλωση που θέλουμε να γίνει ορατή από το main thread (Python). Περισσότερα για αυτή την εντολή αναφέρουμε στην επόμενη ενότητα. Οι ολοκληρωμένες δηλώσεις που περιγράφουν τη διαδικασία αυτή είναι:
    - PyRun_SimpleString(" q.put(l1) ")
    - PyRun_SimpleString(" q.put(l2) ")
3. μεταφέρουμε τη ροή εκτέλεσης στο communication thread, μέσο των ελεγκτών που έχουμε δημιουργήσει στο main thread με τις δηλώσεις:
    - PyRun_SimpleString(" docomputations=false ")
    - PyRun_SimpleString(" docommunication=true ")
    - PyRun_SimpleString(" wait until docomputation == true ")

    Με την πρώτη θέτουμε τον ελεγκτή, μέσω του οποίου επιτρέπουμε στο computation thread να εκτελέσει επόμενο βήμα υπολογισμών, με false, ώστε να τους διακόψουμε. Με την επόμενη εκκινούμε την λειτουργία του communication thread, θέτοντας τον αντίστοιχο ελεγκτή με true, ενώ με την τρίτη ενημερώνουμε το computation thread

ότι θα πρέπει να περιμένει μέχρι κάποια άλλη μέθοδος να θέσει τον ελεγκτή του ξανά σε τιμή true.

4. αφού επιτρέψουμε ξανά τη λειτουργία στο computation thread, εισάγουμε τις νέες τιμές , l1 και l2, που λάβαμε από την ουρά q στις θέσεις που αντιστοιχούν στις συνοριακές τιμές του επεξεργαστή. Η ακολουθία εντολών που αντιπροσωπεύουν τα όσα περιγράψαμε στο βήμα αυτό είναι:
    - PyRun_SimpleString(" l1=q.get() ")
    - PyRun_SimpleString(" l2=q.get() ")
    - solutionTable[valueOfFirstLineAtTheCurrentTime]=l1
    - solutionTable[valueOfLastLineAtTheCurrentTime]=l2

Η διαδικασία που μόλις περιγράψαμε αποτελεί την επέκταση που δέχθηκε η Mol-C, ώστε να ανταποκριθεί στις απαιτήσεις της παράλληλης υλοποίησής μας.

3.3.1.4 Συνάρτηση communication του each_node.py

Η μέθοδος communication που ορίζουμε στο each_node.py απευθύνεται στο communication thread. Με την παραπάνω μέθοδο επιτελούμε τη δουλειά που πρέπει να διεκπεραιώσουμε μέσω του communication thread. Όπως περιγράψαμε στο βήμα 3 της προηγούμενης υπό-ενότητας, τη διαχείριση του επεξεργαστή την έχουμε μεταφέρει στο communication thread.

Αυτά που πρέπει να εκπληρώσουμε στο εν λόγω thread, τη χρονική περίοδο που δεσμεύουμε την υπολογιστική δύναμη για αυτό, τα περιγράφουμε στα επόμενα στάδια, που τα συνοδεύουμε και από τις αντίστοιχες εντολές όπως εμφανίζονται στο Σχήμα 3.3.1 :

1. παίρνουμε από την ουρά q τις τιμές που εισάγαμε από το computation thread και να τις αναθέτουμε σε δύο μεταβλητές, l1, l2, και έπειτα, τις μετατρέπουμε σε string (αυτό γίνεται για να μην έχουμε απώλεια ακριβείας σε δεκαδικά ψηφία, λόγο στρογγυλοποίησης) :
    - l1=q.get()
    - l2= q.get()
    - data1=string(l1)
    - data2=string(l2)
2. υπολογίζουμε με βάσει το μοναδικό αναγνωριστικό του (myId, στην προκειμένη περίπτωση 3) ποιοι επεξεργαστές είναι γειτονικοί του:

- upProcessor = myId+1
- downProcessor = myId -1

3. στέλνουμε τα δεδομένα που απευθύνονται σε κάθε ένα από τους γειτονικούς του επεξεργαστές και λαμβάνουμε από αυτούς, τις τιμές που πρέπει να επιτρέψουμε στο computation thread. Εδώ χρησιμοποιούμε ως μέσο επικοινωνίας το Message Passing Interface, το οποίο παρουσιάζουμε αναλυτικότερα σε ενότητα που ακολουθεί. Είναι σκόπιμο να αναφέρουμε ότι η συνάρτηση αποστολής ενός μηνύματος επικοινωνίας από έναν επεξεργαστή σε έναν άλλο είναι η sent, και η αντίστοιχη λήψης μηνύματος η recieveString. Συνεπώς, η λειτουργία αυτή γίνεται μέσω των εντολών :
    - send(data1, downProcessor)
    - send(data2, upProcessor)
    - recieveString(upProcessor, data2)
    - receiveString(downProcessor, data1)
4. τις τιμές που λαμβάνουμε από τους γειτονικούς επεξεργαστές, τις περνάμε με την μορφή αριθμών στην ουρά, ώστε το computation thread να μπορεί να τις παραλάβει και να τις χρησιμοποιήσει:
    - l1=[data1]
    - l2=[data2]
    - q.put(l1)
    - q.put(l2)
5. θέτουμε τον ελεγκτή του computation thread ως true, για να ενεργοποιήσουμε το ομώνυμο thread και σταματάμε τη λειτουργία του τρέχοντος κάνοντας το docommunication = false, μέχρι να ζητηθεί από το computation thread να επικοινωνήσει ξανά με τους γειτονικούς επεξεργαστές.

Σε αυτή την ενότητα που μόλις παρουσιάσαμε, περιγράψαμε λεπτομερώς θέματα της διπλωματικής που έχουν τεχνικό ή αλγοριθμικό ενδιαφέρον. Παρόλα αυτά, δεν αναφέραμε τα εργαλεία και τις βιβλιοθήκες που χρησιμοποιούμε, για να υλοποιήσουμε την παράλληλη εφαρμογή της μεθόδου των γραμμών. Αυτό το κάνουμε στην ενότητα που ακολουθεί.

3.3.2 Πλατφόρμες και προγραμματιστικά εργαλεία

Σε αυτή την ενότητα περιγράφουμε τα χαρακτηριστικά της συγκεκριμένης υλοποίησης, όπως είναι η πλατφόρμα ανάπτυξης και εκτέλεσης, τα προγραμματιστικά εργαλεία και οι απαιτήσεις της εφαρμογής σε hardware.

Για τις ανάγκες αναπτύξεως αυτής της διπλωματικής χρησιμοποιήσαμε το λειτουργικό περιβάλλον Linux. Το επιλέξαμε, πρώτον γιατί είναι το λειτουργικό σύστημα, το οποίο υποστηρίζει τα παράλληλα υπολογιστικά συστήματα, cluster, όπως το Centaurus του τμήματος, στο οποίο προσανατολιζόμαστε να εκτελέσουμε την παράλληλη εφαρμογή. Επιπρόσθετα, στις εκδόσεις του Linux βρίσκουμε προ-εγκατεστημένες βιβλιοθήκες και editors γλωσσών προγραμματισμού όπως της C και της Python, που μας καθιστούν ικανούς να κάνουμε άμεση ανάπτυξη και εκτέλεση της εφαρμογής. Επιπλέον, το λειτουργικό αυτό, μας δίνει την δυνατότητα μέσω της κονσόλας και της πληθώρας των εντολών της, που καλύπτουν κάθε προγραμματιστική ανάγκη να ξεκινήσουμε άμεσα την υλοποίηση, γλιτώνοντάς μας από την διαδικασία επιλογής κάποιας πλατφόρμας ανάπτυξης. Τα παραπάνω σε συνδυασμό με το γεγονός ότι όλες του οι εκδόσεις είναι συμβατές μεταξύ τους και υπό την μορφή open source, μαζί με όλες τις επεκτάσεις τους, μας ωθούν στο να το χρησιμοποιήσουμε.

Για την παράλληλη εκτέλεση επιλέξαμε το cluster του τμήματος, **Centaurus**. Όπως έχουμε αναφέρει, cluster είναι ο ευρέως διαδεδομένος όρος των ανεξαρτήτων υπολογιστών που συνδυάζονται σε ένα ενιαίο σύστημα μέσω του λειτουργικού συστήματος, ελεύθερης μορφής, Linux και της δικτύωσης. Ομώνυμα υπολογιστικά συστήματα, χρησιμοποιούνται σε εφαρμογές που απαιτούν είτε Υψηλή Διαθεσιμότητα, είτε Υψηλή Απόδοση Υπολογισμών είτε και τα δύο γιατί παρέχουν μεγαλύτερη αξιοπιστία και υπολογιστική δύναμη από ότι ένας υπολογιστής μπορεί να προσφέρει.

Παρακάτω παραθέτουμε α χαρακτηριστικά του centaurus, του cluster τμήματος:

- 5 κόμβοι
- Κάθε κόμβος 2 επεξεργαστές Dual Core Xeon 2.0 Ghz
- 1 GB RAM
- 80 GB ULTRA SCSI μονάδα αποθήκευσης
- Τοπικό δίκτυο 1 Gbit

Ακόμα το λογισμικό που είναι περασμένο τόσο πάνω στον κεντρικό κόμβο όσο και στους υπόλοιπους κόμβους είναι :

- Suse Linux 10.1
- Για επικοινωνία έχει χρησιμοποιηθεί το LAM-MPI, υλοποίηση του πρότυποπου Message Passing Interface.
- Για διαχείριση και ανάθεση διεργασιών χρησιμοποιεί το λογισμικό Tornado OpenPBS (Open Portable Batch System).

Στη συνέχεια αυτής της ενότητας και στα κεφάλαια που ακολουθούν, παρουσιάζουμε την γλώσσα προγραμματισμού που χρησιμοποιούμε για της ανάγκες ανάπτυξης της παράλληλης εφαρμογής της μεθόδου των γραμμών. Εξηγούμε τις αιτίες που μας οδήγησαν στην επιλογή της γλώσσας προγραμματισμού υλοποίησης, όπως και τα εργαλεία αυτής που χρησιμοποιήσαμε για να επιτύχουμε τον απώτερο σκοπό.

3.3.2.1 Python

Για τις ανάγκες αυτής της διπλωματικής χρησιμοποιούμε την γλώσσα προγραμματισμού Python. Η Python είναι μια υψηλού επιπέδου interpreted γλώσσα προγραμματισμού. Αυτό σημαίνει ότι η μετάφραση των εντολών γίνεται από τον διερμηνέα (interpreter), κατά τον χρόνο εκτέλεσης. Τα πλεονεκτήματα της Python ([12]) είναι:

- Η ευκολία στην εκμάθηση. Αυτό συμβαίνει γιατί είναι υψηλού επιπέδου γλώσσα, κοντά στη φυσική γλώσσα του ανθρώπου.
- Εύκολη στη χρήση και στο debugging, καθώς ο interpreter αναλύει τις εντολές ενώ αυτές εκτελούνται, παρέχοντας στο προγραμματιστή πολύ καλή εποπτεία του κώδικα.
- Ανεπτυγμένη επιστημονική κοινότητα. Αυτό μεταφράζεται σε πλήρης υποστήριξη για την ανάπτυξη εφαρμογών, μέσω mailing lists για πληθώρα εφαρμογών της (όπως http://mail.python.org/mailman/listinfo/python-committers) και forums (όπως www.thescripts.com/forum) στις οποίες μπορεί να αναρτηθεί κάποια απορία-προβληματισμός και σε σύντομο χρονικό διάστημα να απαντηθεί.
- Open source, που σημαίνει ότι οι εφαρμογές, βιβλιοθήκες και modules(αυτοτελείς μονάδες προγράμματος, όπως Scientific Python, Numerc, Numarray) που ανατάσσονται, είναι ελεύθερες προς το κοινό μαζί με τον κώδικά τους, δίνοντας την δυνατότητα να τις εντάξει κάποιος απευθείας στην εφαρμογή του, όπως και να επέμβει σε αυτές.

- Επεκτάσιμη, καθώς με την χρήση open source εργαλείων όπως το SWIG, f2py δημιουργούνται εύκολα wrappers σε modules, που είναι γραμμένες σε C, C++, Fortran.
- Αλληλεπιδραστική, με γλώσσες προγραμματισμού όπως η C , διότι στην πρότυπη βιβλιοθήκη βρίσκεται και η ενότητα C-Python API η οποία επιτρέπει την Python να επεμβαίνει στον κώδικα της C.

Τα παραπάνω πλεονεκτήματα αποτέλεσαν αιτίες που οδήγησαν την Python να είναι μέσα στις υποψήφιες γλώσσες ανάπτυξης της παράλληλης μεθόδου των γραμμών. Ωστόσο, τα τρία τελευταία σημεία είναι αυτά που μας έκαναν να την προτιμήσουμε. Αυτό συμβαίνει γιατί μας δίνει την δυνατότητα να κάνουμε χρήση του ήδη υπάρχοντος κώδικα της σειριακής υλοποίησης της μεθόδου, μετατρέποντάς τον μέσω του εργαλείου SWIG. Επίσης, μας επιτρέπει να επέμβουμε στον Mol-C κώδικα με εντολές γραμμένες σε Python (C-Python API, συνάρτηση PyRun_SimpleString), για να επιτύχουμε την εναλλαγή και επικοινωνία των threads communication και computation. Ενώ ακόμα υποστηρίζει το MPI (Message Passing Interface, μέσω του module Scientific) που χρησιμοποιείται για την επικοινωνία μεταξύ των επεξεργαστών του cluster της σχολής, όπως και ταυτόχρονο προγραμματισμό μέσω της ανεπτυγμένης κλάσης threading.

Η εναλλακτική πρόταση που υπήρχε, για υλοποίηση της διπλωματικής ήταν το λογισμικό Matlab ([12]). Αυτή η σκέψη ήταν εύλογη, από την στιγμή που το Matlab παρέχει διεπαφές που χρησιμοποιούν τις υψηλότερου επιπέδου ανεπτυγμένες βιβλιοθήκες για να εκτελέσουν ευαίσθητες υπολογιστικές λειτουργίες, όπως ο πολλαπλασιασμός και η παραγοντοποίηση πινάκων. Το Matlab εξάλλου είναι πολύ διαδεδομένο για ανάπτυξη μαθηματικών αλγορίθμων τις επιστήμης υπολογιστών.

Επιπλέον, το Matlab ταιριάζει στις ανάγκες της παράλληλης εφαρμογής μας, καθώς εκτελεί κώδικα C, μέσω του C-Matlab Interface και υποστηρίζει εργαλεία ανάπτυξης παράλληλων εφαρμογών όπως το Parallel Toolbox και το pMatlab project. Ωστόσο, το γεγονός ότι το Matlab και οι εφαρμογές του δεν είναι open source, μας απέτρεψε από τη χρήση του ως λογισμικό ανάπτυξης. Για την μετατροπή της ανεπτυγμένης Mol-C εφαρμογής, σε μορφή που να επιτρέπει την ενσωμάτωση της σε εφαρμογές υλοποιημένες σε Python και συγκεκριμένα στον παράλληλο αλγόριθμο της μεθόδου των γραμμών, κάνουμε χρήση του αυτοματοποιημένου εργαλείου SWIG.

3.3.2.2 SWIG

Το **Swig (Simple Wrapper and Interface Generator,** [13]) είναι ένα εργαλείο που μας επιτρέπει την πρόσβαση σε κώδικα της C από την Python. Με το SWIG αυτόματα παράγουμε ολοκληρωμένη διεπαφή Python για ήδη υπάρχον κώδικα της C, όπως είναι και η Mol-C . Στη χρήση αυτού του εργαλείου οδηγηθήκαμε γιατί θέλουμε να επωφεληθούμε των δυνατοτήτων που παρέχει ο προγραμματισμός σε C που είναι:

- Απόλυτη υποστήριξη από βιβλιοθήκες για τις ανάγκες προγραμματισμού
- Υψηλή απόδοση
- Μεγάλη κοινότητα χρηστών

Προτού περιγράψουμε την διαδικασία που ακολουθούμε για να χρησιμοποιήσουμε από την Python, συναρτήσεις που έχουν αναπτυχθεί σε C, πρέπει να αναφέρουμε ότι σε περιβάλλον Linux που δουλεύουμε, το πακέτο του SWIG-Python είναι προ-εγκατεστημένο και άρα έτοιμο για χρήση. Για να κάνουμε ευκολότερη την ανάλυση της διαδικασίας, θα την παρακολουθήσουμε μέσω ενός παραδείγματος. Έστω ότι αποσκοπούμε στην χρήση της fact , my_mod συνάρτησης και της καθολικής μεταβλητής My_variable που τις έχουμε ορίσει σε ένα αρχείο example.c.

```
 /* example.c */
double My_variable = 3.5;

/*Compute factorial of n*/
int fact(int n) {
     if (n <= 1) return 1;
     else return n*fact(n-1);
}

/* Compute n mod m*/
int my_mod(int n, int m){
      return(n%m);
}
```

Για να γίνει αυτό πρέπει να δημιουργήσουμε ένα **αρχείο διεπαφής**, που να παραθέτουμε τις C ANSI δηλώσεις των πραγμάτων που θέλουμε να αποκτήσουμε πρόσβαση, όπως δηλώσεις μεταβλητών, συναρτήσεων, δομών δεδομένων. Οτιδήποτε είναι περασμένο μέσα στα %{ - %} αντιγράφεται αυτούσιο στον wrapper κώδικα που δημιουργείται από το SWIG. Συνήθως σε αυτό το σημείο εισάγονται τα header files, οι κεφαλίδες συναρτήσεων και κώδικας υποστήριξης ειδικών λειτουργιών, όπως το πέρασμά αρχείων εισόδου της C main. Στο συγκεκριμένο παράδειγμα, το αρχείο διεπαφής θα έχει την ακόλουθη μορφή:

```
// example.i
%module example
%{
/*Put headers here*/
extern double My_variable;
extern int    fact(int);
extern int    my_mod(int n, int m);
%}

/*Put C declarations here*/
extern double My_variable;
extern int    fact(int);
extern int    my_mod(int n, int m);
```

Το example στη δήλωση %module example, προσδιορίζει το όνομα του module που δημιουργούμε με το SWIG. Στη συνέχεια, πρέπει να κατασκευάσει το SWIG, μια διεπαφή της Python με τον κώδικα της C. Αυτό γίνεται με την ακολουθία των εντολών :

```
linux console/.appropriate_path>swig -python example.i        (3.3.2.2)
linux console/.appropriate_path>gcc -c -fpic example.c
      example_wrap.c -I/usr/local/include/python2.5
linux console/.appropriate_path>gcc -shared example.o
      example_wrap.o -o example.so
```

Με την πρώτη εντολή, το SWIG παράγει ένα αρχείο 'example_wrap.c', που δίνει όλη την απαραίτητη πληροφορία στην Python για το πώς να χειριστεί τις δηλώσεις της C και πρέπει να μεταφραστεί μαζί με το αρχείο example.c. Έπειτα το Python module, 'example.so' με την μορφή βιβλιοθήκης μπορεί να φορτωθεί δυναμικά. Όπως συμβαίνει με κάθε Python module, μονάδες του οποίου επιθυμούμε να χρησιμοποιήσουμε σε μία Python εφαρμογή, το όνομα του module που παράγουμε, το εισάγουμε χωρίς την επέκταση του αρχείου, στην εφαρμογή μας, μέσω της δήλωσης import name. Έτσι μπορούμε να προσπελάσουμε τις καθολικές μεταβλητές που ορίζονται σε αυτό, να τις χρησιμοποιήσουμε για υπολογισμούς και να κάνουμε χρήση των μεθόδων που περιέχονται σε αυτό. Αυτό γίνεται με τον ακόλουθο τρόπο:

```
linux console/.appropriate_path>python
>>>import example
>>>example.fact(4)
24
>>>example.my_mod(23,7)
2
>>>example.cvar.My_variable +4.5
7.5
```

Ή σε μορφή απλού προγράμματος:

```
#test_example.py
import _example
x=_example.fact(4)
y=_example.my_mod(23,7)
l=_example.cvar.My_variable +4.5
```

```
print x,y,l

linux console/.appropriate_path>python test_python.py
```

Στη δική μας περίπτωση, για να χρησιμοποιήσουμε συναρτήσεις και μεταβλητές της Mol-C αναπτύξαμε ένα αρχείο interface.i, στο οποίο περνάμε όλες τις δηλώσεις συναρτήσεων, δομών δεδομένων και καθολικών μεταβλητών, που υπάρχουν στην Mol-C εφαρμογή. Επιπλέον, για να εκτελέσουμε αυτούσια την Mol-C εφαρμογή, από το computation thread καλώντας την main(), που βρίσκεται στο αρχείο test_method_of_lines_1d.c, στο interface.i προσθέτουμε την ακόλουθη οδηγία για το πέρασμα ορισμάτων στη main(), που βρέθηκε έτοιμη στο δίκτυο:

```
%typemap(in) char ** {
  /* Check if is a list */
  if (PyList_Check($input)) {
    int size = PyList_Size($input);
    int i = 0;
    $1 = (char **) malloc((size+1)*sizeof(char *));
    for (i = 0; i < size; i++) {
      PyObject *o = PyList_GetItem($input,i);
      if (PyString_Check(o))
        $1[i] = PyString_AsString(PyList_GetItem($input,i));
      else {
        PyErr_SetString(PyExc_TypeError,"list must contain strings");
        free($1);
        return NULL;
      }
    }
    $1[i] = 0;
  } else {
    PyErr_SetString(PyExc_TypeError,"not a list");
    return NULL;
  }
}

%typemap(freearg) char ** {
  free((char *) $1);
}
// Now a test function
%inline %{
int print_args(char **argv) {
    int i = 0;
    while (argv[i]) {
         printf("argv[%d] = %s\n", i,argv[i]);
         i++;
    }
    return i;
}
```

Εκτελούμε το παραπάνω σύνολο εντολών (3.3.2.2) στην κονσόλα του συστήματος με την προσθήκη της –Dmain=oldmain για να μετονομάσουμε την main() συνάρτηση του test_method_of_lines_1d.c σε κάτι άλλο (oldmain), καθώς το όνομα είναι δεσμευμένο στην

Python και χρησιμοποιείται στο each_node.py για να προσδιορίσει στον interpreter την αρχή της εκτελέσεως.

```
linux console/.appropriate_path>swig -python interface.i
linux console/.appropriate_path>gcc -c -fpic *.c
      Dmain=oldmain I/usr/local/include/python2.5
linux console/.appropriate_path>gcc -shared *.o -o interface.so
```

Έπειτα από τα παραπάνω, έχουμε στην διάθεση μας το module _interface.so, το οποίο χρησιμοποιούμε για τους υπολογισμούς, στο computation thread. Παρόλα αυτά, στον κώδικα της Mol-C που μετατρέψαμε σε module της Python, εισάγουμε και κάποιες εντολές που να είναι αναγνωρίσιμες από την Python. Για αυτό το σκοπό χρησιμοποιήσαμε το Python/C API.

3.3.2.3 Python/C API

Όπως έχουμε αναφέρει, μια από τις αιτίες που μας οδήγησαν στην επιλογή της Python, ως γλώσσας ανάπτυξης της παράλληλης εφαρμογής της μεθόδου των γραμμών, είναι η αλληλεπίδραση της με την C. Η εφαρμογή διεπαφής προγραμματιστή (API – Application Programmer Interface [14]) της Python, παρέχει στους προγραμματιστές της C, πρόσβαση στον Python interpreter σε διάφορα επίπεδα. Υπάρχουν δύο λόγοι που μπορούν να οδηγήσουν κάποιον στην χρήση του **Python/C API**:

1. Να αποσκοπεί σε μονάδες επέκτασης (extension modules) του Python διερμηνέα, με C modules. Τη δυνατότητα επέκτασης όπως είδαμε στην ενότητα 3.3.2.2 SWIG την πράξαμε με το ομώνυμο εργαλείο ανάπτυξης μονάδων επέκτασης της Python.
2. Να χρειάζεται να χρησιμοποιήσει την Python ως ένα συστατικό σε μία μεγαλύτερη εφαρμογή. Αυτή η τεχνική αναφέρεται ως ενσωμάτωση της Python στην εφαρμογή.

Τη Python/C διεπαφή την χρησιμοποιήσαμε για τον δεύτερο από τους δύο λόγους. Αυτό κρίθηκε αναγκαίο, καθώς στον σειριακό κώδικα Mol-C θέλουμε:

- Να αποθηκεύουμε σε λίστα της Python, τις συνοριακές τιμές κάθε επεξεργαστή, έπειτα από κάθε υπολογιστικό βήμα
- Να διακόπτουμε την ροή εκτέλεσης των υπολογισμών, του computation thread, εκκινώντας ξανά το thread της επικοινωνίας, για την ανταλλαγή δεδομένων μεταξύ των επεξεργαστών.

Από το Python/C API κάναμε χρήση της πρότυπης συνάρτησης **PyRun_SimpleString**, καθώς και των συναρτήσεων **Py_Initialize** και **Py_Finalize** για την αρχικοποίηση και τον τερματισμό ενεργοποίησης του interpreter της Python, από την C. Η δήλωση-ανάλυση της συνάρτησης αυτής ακολουθεί:

> ***int PyRun_SimpleString (char * command)****:εκτελεί τον κώδικα Python που προσδιορίζει το όρισμα command, στο __main__ μέρος του προγράμματος. Αν __main__ μέρος δεν υφίσταται, τότε δημιουργείται. Επιστρέφει 0 όταν η εντολή εκτελεστεί με επιτυχία και 1 στην περίπτωση που συμβεί κάποια εξαίρεση.*

Στη δικιά μας εφαρμογή υπάρχει η __main__ στο each_node.py, το οποίο όπως έχουμε αναφέρει στην ενότητα 3.3.1.2 εκκινεί την παράλληλη εκτέλεση και δημιουργεί το main_thread. Συνεπώς, επιλέγουμε τη συγκεκριμένη εντολή, γιατί αποθηκεύει τιμές, από τη συνάρτηση MasterCalls του αρχείου class_method_of_lines_1d.c, στη λίστα που βρίσκεται στο main thread και δίνει σε αυτό την δυνατότητα εναλλαγής μεταξύ των δύο threads.

3.3.2.4 Scientific Python –MPI

Στο cluster, για την επικοινωνία μεταξύ επεξεργαστών, χρησιμοποιούμε το μοντέλο επικοινωνίας MP (Message Passing). Σε αυτό το μοντέλο επικοινωνίας, οι μονάδες επεξεργασίας επικοινωνούν μεταξύ τους, με ανταλλαγή μηνυμάτων. Οι message passing παράλληλες πλατφόρμες επιτρέπουν την εκτέλεση διαφορετικών προγραμμάτων από κάθε μονάδα επεξεργασίας. Οι βασικές εντολές που χρησιμοποιούν τέτοιες πλατφόρμες είναι :

- **send, receive** : για αποστολή και λήψη δεδομένων μεταξύ των επεξεργαστών
- **μέθοδοι ταυτοποίησης επεξεργαστών**: για να είναι σε θέση οι επεξεργαστές να αντιλαμβάνονται κατά πόσο τα δεδομένα που έλαβαν προέρχονται από τον εξουσιοδοτημένο επεξεργαστή και αντίστοιχα σε ποιόν επεξεργαστή πρέπει να αποστείλουν τα δεδομένα τους.

Για αυτές τις παράλληλες πλατφόρμες έχουν αναπτυχθεί διεπαφές οι οποίες διευκολύνουν τις διαδικασίες επικοινωνίας και ονομάζονται **Message Passing Interfaces** (MPIs). Για την Python έχουν αναπτυχθεί πολλά modules που εκτελούν το συγκεκριμένο μοντέλο επικοινωνίας. Μερικά από αυτά είναι το mpi4py, PyPar, PyMPI κ.α. Εμείς επιλέξαμε να κάνουμε χρήση του Scientific module, που υποστηρίζει τις απαραίτητες MPI λειτουργίες, με on-line παραδείγματα, ενώ παρέχει documentations για την περιγραφή και τον τρόπο χρήσης των συναρτήσεων του.

Όπως αναφέραμε σε προηγούμενη ενότητα, για να εκτελέσουμε την παράλληλη εφαρμογή που έχουμε αναπτύξει, δίνουμε στη γραμμή εντολών του cluster την εντολή:

mpirun –np num_of_processors mpipython each_node.py

Με αυτή την εντολή ζητάμε από το παράλληλο υπολογιστικό μας περιβάλλον, να εκτελέσει την each_node.py παράλληλη διεργασία, σε κάθε έναν από τους num_of_processors επεξεργαστές, χρησιμοποιώντας ως μέθοδο επικοινωνίας το MP. Την πληροφορία των χαρακτηριστικών του κόσμου επικοινωνίας, που απαρτίζεται από τους num_of_processors επεξεργαστές, πρέπει να την μεταφέρουμε σε κάθε παράλληλη διεργασία. Τη λειτουργία αυτή την διεκπεραιώνουμε με τη χρήση της **ScientificMPI** διεπαφής, του ScientificPython module. Αυτή μας παρέχει, μέσω της κλάσης **Scientific.MPI.world**, έναν πληροφοριοδότη του κόσμου επικοινωνίας,. Για αυτό το λόγο, δημιουργούμε ένα αντικείμενο αυτής της κλάσης :

communicator = MPI.world.duplicate()

Αυτό το αντικείμενο μας δίνει δύο πληροφορίες, σχετικά με τον κόσμο επικοινωνίας, σε κάθε διεργασία :

- **communicator.size** : αναφέρεται στο πλήθος των διαθέσιμων επεξεργαστών που αποτελούν τον κόσμο του communicator.
- **communicator.rank**: επιστρέφει το αποκλειστικό αναγνωριστικό του επεξεργαστή, στον οποίο εκτελείται αυτή η εντολή.

Ενώ οι συναρτήσεις που μας παρέχει είναι οι εξής:

- **communicator.send(data, id_of_destination_processor, number)**: για αποστολή δεδομένων. Στέλνει το δεδομένο data, στο id_of_dastination_processor, δίνοντας έναν αριθμό που χαρακτηρίζει την αποστολή
- **data, source, tag=communicator.recieveString(id_of_sender_processor, number)**: για παραλαβή δεδομένων. Αποθηκεύει το string που λαμβάνει, στη μεταβλητή data, το αποκλειστικό αναγνωριστικό του επεξεργαστή αποστολής στη μεταβλητή source, και την τιμή που χαρακτηρίζει την συναλλαγή στην μεταβλητή tag.

3.3.2.5 Thread-Threading module

Για την κατασκευή των thread, επικοινωνίας και υπολογισμών, βασιστήκαμε στην κλάση **Threading** που παρέχει η Python. Η κλάση αυτή είναι υποκλάση της **thread**. Λόγο κληρονομικότητας, παρέχει όλες τις συναρτήσεις της υπέρ-κλάσης thread, αλλά και επιπρόσθετες, δίνοντας μία υψηλού επιπέδου διεπαφή για ταυτόχρονο προγραμματισμό, μέσω threads.

Τα νήματα που δημιουργούμε είναι της κλάσης thread και έχουν την ακόλουθη μορφή:

- **name_of_thread= Thread (target=name_of_function, arg = (list of arguments))**: όπου name_of_function αναφερόμαστε στη λειτουργία – συνάρτηση που πραγματοποιεί το thread με όνομα, name_of_Thread, που δημιουργούμε. Στη λίστα των παραμέτρων εισάγουμε όσες μεταβλητές χρειάζεται η συνάρτηση λειτουργίας του thread.

Από την κλάση αυτή χρησιμοποιούμε τις ακόλουθες συναρτήσεις:

- **Event ()**: αυτή η συνάρτηση επιστρέφει ένα νέο event αντικείμενο. Το αντικείμενο αυτό διαχειρίζεται ένα flag που μπορεί να τεθεί ως true με τη βοήθεια της μεθόδου *set()* και false μέσο της μεθόδου *clear()*. Η συνάρτηση *wait()* μπλοκάρει το αντικείμενο αυτό, έως ότου μια άλλη λειτουργία θέσει το flag που διαχειρίζεται το αντικείμενο, ως true.
- **start ()**: θέτει ένα αντικείμενο της κλάσης threading ενεργό
- **join ()** : συνάρτηση που περιμένει μέχρι το thread που την καλεί να τερματιστεί.

3.3.2.6 Queue module

Το **Queue module** υλοποιεί μία ουρά FIFO, πολύ χρήσιμη στον προγραμματισμό νημάτων όταν η πληροφορία πρέπει να ανταλλαχτεί μεταξύ αυτών, όπως και στην περίπτωσή μας. Η κλάση Queue του module αυτού υλοποιεί όλη την απαραίτητη προστατευμένη σημασιολογία. Στην εφαρμογή μας έχουμε χρησιμοποιήσει αυτή την κλάση, δημιουργώντας μία ουρά μεγέθους δύο στοιχείων. Αυτά τα δύο στοιχεία είναι οι συνοριακές τιμές που έχουν υπολογιστεί στο τέλος κάθε υπολογιστικού βήματος από κάθε επεξεργαστή.

```
q = Queue.Queue( 2 )
```

Για να εισάγουμε και να εξάγουμε στοιχεία από το αντικείμενο q της Queue, εκμεταλλευόμαστε τις συναρτήσεις :

- **put(item)** : η οποία εισάγει το στοιχείο item στην ουρά
- **get ()** : η οποία εξάγει το πρώτο στοιχείο της ουράς.

3.3.2.7 Σύνοψη - Διαδικασία εκτέλεσης

Για να γίνει η εκτέλεση της παράλληλης εφαρμογής της μεθόδου των γραμμών πράξη, ακολουθούμε τα επόμενα βήματα:

- Μεταφέρουμε το φάκελο εφαρμογής στο cluster
- Σηκώνουμε τους κόμβους του cluster:

  linux console>lamboot nodes

  όπου nodes ένα script που ενημερώνει ποιοί κόμβοι αποτελούν το cluster
- Δημιουργούμε ένα φάκελο με το όνομα *shmeia*, στην περίπτωση που δεν υπάρχει. Στο φάκελο αυτό με την ολοκλήρωση της εκτέλεσης της εφαρμογής εμφανίζονται αρχεία με το όνομα *pointsL_i.txt*, όπου L το πλήθος των γραμμών κάθε επεξεργαστή και i ο αριθμός του κάθε επεξεργαστή. Στα αρχεία αυτά περιέχομε πληροφορία σχετικά με τα σημεία, στα οποία εφαρμόσαμε τη μέθοδο των γραμμών στο τελευταίο βήμα υπολογισμού, και την τιμή που υπολογίστηκε σε αυτά. Τα αρχεία δημιουργήθηκαν με σκοπό την επιβεβαίωση της ορθότητας της επίλυσης.
- Μπαίνουμε στο φάκελο txt_input_files (> cd txt_input_files) αντιγράφουμε το αρχείο params_L.txt στον εξωτερικό φάκελο (> cp params_L.txt ../), με L το πλήθος των γραμμών στο οποίο θέλουμε να διακριτοποιηθεί το πεδίο μας.
- Εκτελούμε την ακολουθία εντολών :

```
linux console/.appropriate_path>swig -python interface.i
linux console/.appropriate_path>gcc -fpic -c *.c
       -Dmain=oldmain -I/usr/include/python2.4
linux console/.appropriate_path>ld -shared *.o -o
_interface.so
```

  Η οποία δεν είναι απαραίτητη αν δεν πρόκειται για την πρώτη εκτέλεση
- Εκτελούμε το αρχείο *center_node.py* που δημιουργεί τα αρχεία εισόδου για το πλήθος των επεξεργαστών που σκοπεύουμε να χρησιμοποιήσουμε στην παράλληλη εφαρμογή μας, με την εντολή:

  linux console/.appropriate_path >python center_node.py

  (Κεφάλαιο 3.3.1.1)

- Πληκτρολογούμε την ακόλουθη εντολή :

  linux console/.appropriate_path>mpirun – np i mpipython each_node.py

  όπου i το πλήθος των επεξεργαστών.

- Αν θελήσουμε να τρέξουμε την παράλληλη εφαρμογή με άλλο αρχείο εισόδου, διαγράφουμε πρώτα το υπάρχον (>rm params_L.txt) και επαναλαμβάνουμε τη διαδικασία, των τεσσάρων προηγούμενων βημάτων, με τη σειρά.

- Με την ολοκλήρωση των εκτελέσεων της παράλληλης εφαρμογής εκτελούμε τις ακόλουθες εντολές για να κατεβάσουμε τους κόμβους και να κλείσουμε την επικοινωνία μας με το cluster:

  linux console> lamhalt

  linux console>close

3.4 Αριθμητικά Αποτελέσματα

Για να διαπιστώσουμε τη χρησιμότητα και τη βελτίωση απόδοσης της παράλληλης εφαρμογής συγκριτικά με τη σειριακή, εκτελούμε μετρήσεις στο παράλληλο περιβάλλον cluster Centaurus, που περιγράψαμε νωρίτερα. Για κάθε ένα από τα πειράματα δημιουργήσαμε τα κατάλληλα αρχεία εισόδου που δέχεται καθένας επεξεργαστής. Αυτά τα αρχεία πρέπει να βρίσκονται στον ίδιο φάκελο στον οποίο βρίσκεται και το αρχείο εκτέλεσης, each_node.py. Στις μετρήσεις που εκτελούμε, μεταβάλλουμε τόσο το πλήθος των επεξεργαστών που λαμβάνουν μέρος στους υπολογισμούς, όσο και το μέγεθος του προβλήματος, θέτοντας διάφορες τιμές ως πλήθος γραμμών.

Το πλήθος των επεξεργαστών, με το οποίο εκτελέσαμε τις μετρήσεις διακυμάνθηκε μεταξύ του πλήθος των: 1, 2, 4, 5, 6, 8, 10, 12, 14, 16 επεξεργαστών.

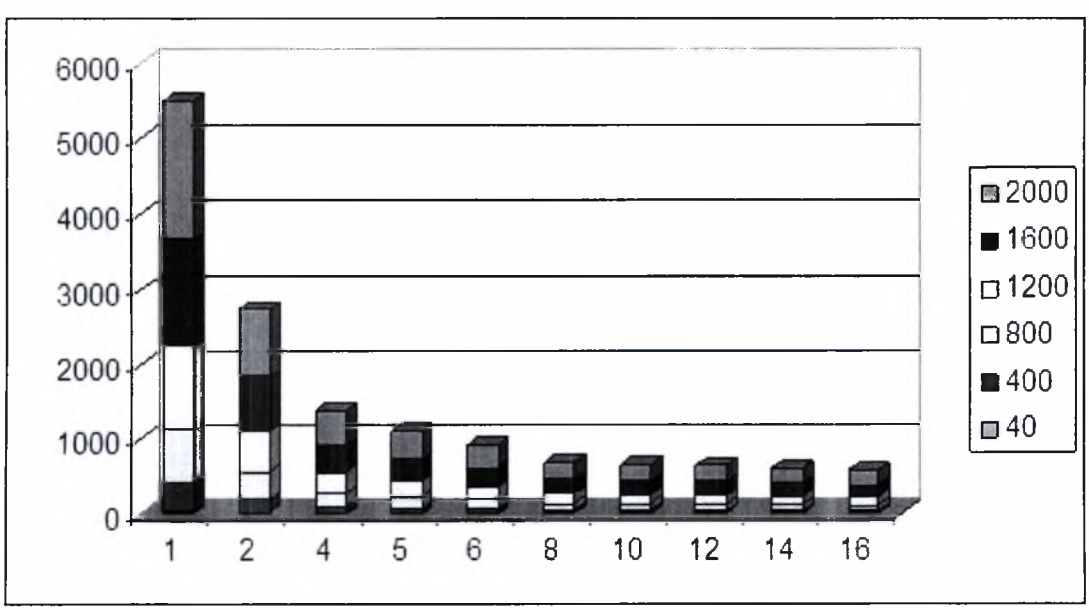
Παράλληλα, το πλήθος των γραμμών στο οποίο διακριτοποιούμε το πεδίο του προβλήματος έλαβε τις ακόλουθες τιμές: 40, 400, 800, 1200, 1600, 2000 γραμμές.

Τα αποτελέσματα που συλλέξαμε από την εκτέλεση των περιπτώσεων αυτών παρουσιάζονται στον ΠΙΝΑΚΑ 1.

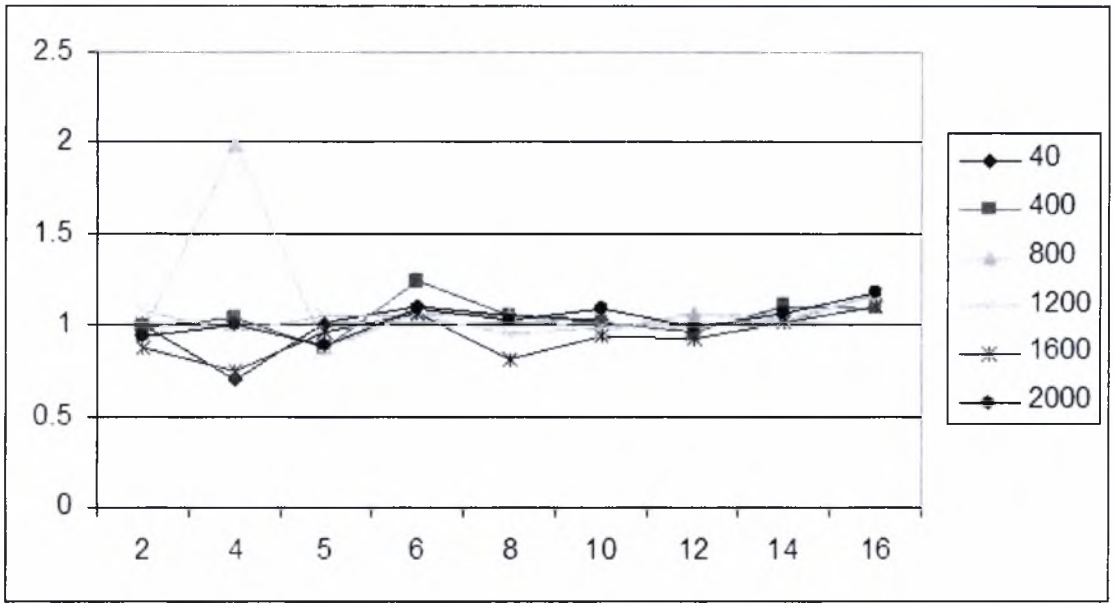
	40		400		800		1200		1600		2000	
Proc	**Comp**	**Comm.**	**Comp**	**Comm.**	**Comp**	**Comm.**	**Comp**	**Comm.**	**Comp**	**Comm.**	**Comp**	**Comm**
1	36.53		363.84		730.28		1094.6		1458.4		1822.2	
2	18.24	1	181.5	0.99	364.37	0.95	546.19	1.08	728.31	0.87	910.16	0.94
4	9.06	0.7	90.14	1.04	181.84	1.99	273.11	0.96	363.69	0.75	455.12	1
5	8.26	1.02	72.88	0.88	146.61	0.87	219.2	1.05	291.17	0.97	363.9	0.89
6	6.39	1.11	60.83	1.25	121.93	1.06	183.04	1.05	243.12	1.07	303.9	1.09
8	4.58	1.05	44.82	1.05	91.01	1.05	136.51	0.95	181.72	0.81	227.42	1.03
10	4.19	1.03	42.33	1.02	85.67	0.97	128.96	0.99	174.43	0.94	217.81	1.09
12			41.65	0.97	84.22	1.07	126.48	1.01	169.74	0.93	210.41	0.99
14			38.72	1.1	78.37	1.03	119.88	1.01	164.77	1.02	205.29	1.07
16			35.61	1.09	75.51	1.17	114.2	1.17	154.99	1.1	201.77	1.18

ΠΙΝΑΚΑΣ 1. Μετρήσεις χρόνου επικοινωνίας και χρόνου υπολογισμών, ανάλογα με το πλήθος των επεξεργαστών και των γραμμών.

Στα Σχήμα 4.3.1 και 4.3.2 που ακολουθούν, παρουσιάζουμε το κόστος επικοινωνίας και υπολογισμών αντίστοιχα, για τις τιμές του ΠΙΝΑΚΑ 1.



Σχήμα 3.4.1 Υπολογιστικό Κόστος



Σχήμα 4.3.2 Επικοινωνιακό Κόστος

ΧΡΟΝΟΒΕΛΤΙΩΣΗ: Ως χρονοβελτίωση δηλώνουμε τον λόγο του χρόνου υπολογισμού της μεθόδου, μιας ποσότητας γραμμών από πολλούς επεξεργαστές, προς το χρόνο υπολογισμού του ίδιου πλήθους γραμμών από έναν μόνο επεξεργαστή.

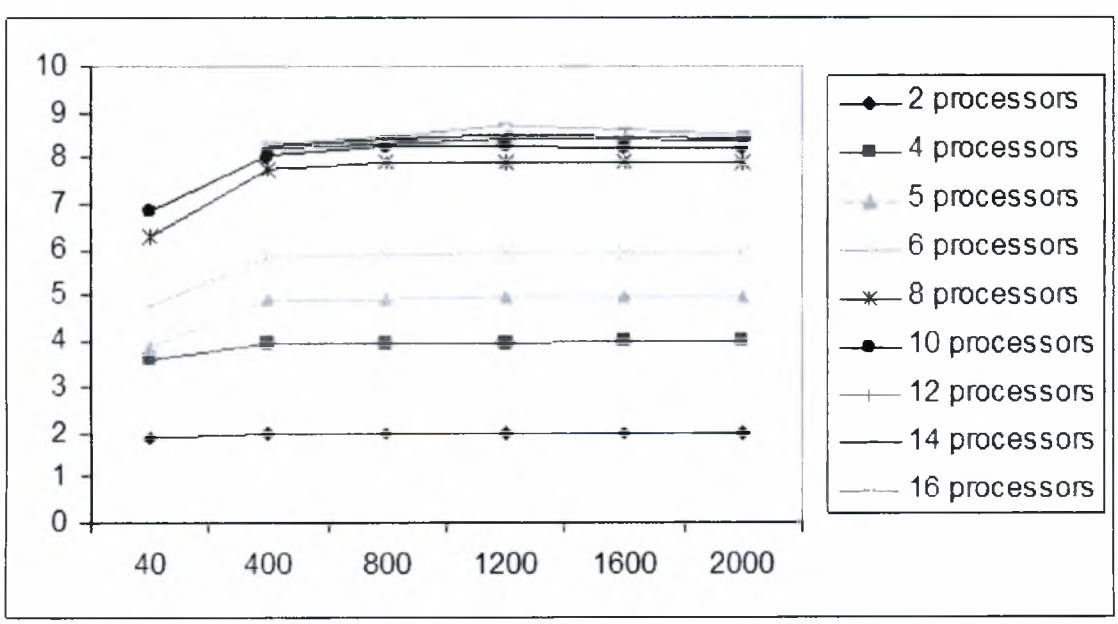
$$speedup = \frac{ComputationTime_{Nprocessors}}{ComputationTime_{Oneprocessor}} \qquad (3.4.1)$$

Από τον ΠΙΝΑΚΑ 1 και την σχέση (3.4.1) δημιουργούμε τον ΠΙΝΑΚΑ 2 της χρονοβελτίωσης που μας δηλώνει πόσος αποδοτική είναι η εφαρμογή μας σε διάφορα πλήθη επεξεργαστών.

	40	400	800	1200	1600	2000
2	1.89569	1.98733	1.99672	1.99664	1.9985063	1.9973365
4	3.59547	3.94792	3.95751	3.98246	3.9907782	3.9895129
5	3.87792	4.91875	4.942	4.95936	4.9813505	4.9811656
6	4.76271	5.81586	5.91272	5.93043	5.9555292	5.9559078
8	6.28744	7.74952	7.90346	7.92044	7.9420029	7.943374
10	6.84082	8.09613	8.28922	8.27597	8.2241583	8.2255676
12		8.20568	8.31754	8.40562	8.4023161	8.3845304
14		8.25408	8.41918	8.52012	8.4563957	8.4069666
16		8.30685	8.48177	8.74754	8.6336135	8.5273527

ΠΙΝΑΚΑΣ 2. Χρονοβελτίωση παράλληλης υλοποίησης συναρτήσει χρόνου εκτέλεσης υπολογισμών σε έναν επεξεργαστή.

Κατά αντιστοιχία με τον ΠΙΝΑΚΑ 1 και τα Σχήματα 3.4.1 και 3.4.2, στο Σχήμα 3.4.3 που ακολουθεί αναπαριστάνουμε τις τιμές της χρονοβελτίωσης που απαρτίζουν τον ΠΙΝΑΚΑ 2.



Σχήμα 3.4.2 Χρονοβελτίωση Αλγορίθμου

Ακολουθούν τα συμπεράσματα που εξάγουμε από την παρατήρηση του Σχήματος 3.4.2:

- Σε κάθε περίπτωση η παράλληλη υλοποίηση που αναπτύξαμε, είναι περισσότερο αποδοτική σε σχέση με την σειριακή.
- Σε κάθε περίπτωση η χρονοβελτίωση που λαμβάνουμε σε παράλληλο περιβάλλον N επεξεργαστών διέπεται από τη σχέση :

  Speed up N= computation cost N/ computation cost 1 < N

  Κοντά όμως στο N.
- Σε κάθε περίπτωση η χρονοβελτίωση που λαμβάνουμε από την εκτέλεση της παράλληλης εφαρμογής μας είναι μεγαλύτερη, όταν την εκτελούμε σε περισσότερους κόμβους του cluster.
- Όταν εκτελούμε την εφαρμογή μας σε περισσότερους των 10 κόμβους, τότε δεν παρατηρούμε βελτιώσεις, ανάλογες με το πλήθος των κόμβων. Το γεγονός αυτό το εναποθέτουμε στα ακόλουθα:
  - Το claster αποτελείται από 5 κόμβους, 2 επεξεργαστών ο κάθε ένας, με αποτέλεσμα ο διαχωρισμός του προβλήματος σε περισσότερες των 10 διεργασίες, να μην προσφέρει ουσιαστική βελτίωση όσον αφορά το χρόνο εκτέλεσης.
  - Κάθε παράλληλος αλγόριθμος έχει ένα οριακό σημείο αποδόσεως. Μετά από αυτό το σημείο, η απόδοση συγκλείνει και τελικά σταθεροποιείται, ακόμα και αν το κλιμακώσουμε σε μεγαλύτερο επίπεδο παραλληλίας (**σημείο κορεσμού - saturation – Amdahl's point**),

4

Επίλογος

Ολοκληρώνοντας την παρουσίαση της διπλωματικής εργασίας, φτάνουμε σε αυτό το τελευταίο κεφάλαιο όπου κάνουμε μια σύντομη ανασκόπηση των όσων παρουσιάσαμε στα προηγούμενα. Εδώ παραθέτουμε και τα συμπεράσματά μας αναφορικά με την εφαρμογή που δημιουργήσαμε, εκτελώντας αυτή την διπλωματική εργασία. Τέλος, δίνουμε και κάποιες ιδέες στον αναγνώστη, για το πώς θα μπορούσε να διευρύνει αυτή την εφαρμογή που υλοποιήσαμε.

4.1 Σύνοψη και συμπεράσματα

Στα προηγούμενα κεφάλαια καλύψαμε τα ζητήματα της διπλωματικής εργασίας μας. Συγκεκριμένα στο πρώτο κεφάλαιο αναφέραμε τις έννοιες της αριθμητικής επίλυσης Μερικών Διαφορικών Εξισώσεων και των Κατανεμημένων Συστημάτων, με τις οποίες ασχολούμαστε στην εργασία, ενώ δώσαμε και μια γενική ιδέα στον αναγνώστη των όσων ακολουθούν στις επόμενες ενότητες.

Στο Δεύτερο κεφάλαιο, παρουσιάσαμε την κλάση διαφορικών εξισώσεων που μας απασχόλησε στην εργασία, και τον αλγόριθμο που χρησιμοποιούμε για να επιλύσουμε προβλήματα της κλάσης αυτής. Η μέθοδος με την οποία επιτελούμε αυτό το σκοπό, είναι η μέθοδος των γραμμών. Στο ίδιο κεφάλαιο περιγράψαμε μια ήδη υλοποιημένη σειριακή

εφαρμογή αυτής της μεθόδου, την Mol-C, πάνω στην οποία δομήσαμε την δική μας παράλληλη υλοποίηση της Mol.

Περνώντας, στο τρίτο κεφάλαιο, που αποτελεί και το κεφάλαιο της ανάπτυξης, παρουσιάσαμε τον σχεδιασμό του παράλληλου αλγορίθμου της μεθόδου των γραμμών. Αναλύσαμε τις κύριες λειτουργίες της παράλληλης υλοποίησης και τα σημεία που εμφανίζουν προγραμματιστικό ενδιαφέρον. Στη συνέχεια, δώσαμε τα προγραμματιστικά εργαλεία, τις μεθόδους και τα χαρακτηριστικά της πλατφόρμας που χρησιμοποιήσαμε, για να πετύχουμε τον στόχο μας. Κλείνοντας αυτό το κεφάλαιο, παραθέσαμε τα αριθμητικά αποτελέσματα που λάβαμε από την εκτέλεση της εφαρμογής μας.

Από τα αποτελέσματα της εφαρμογή μας, εκτελώντας την στο παράλληλο υπολογιστικό περιβάλλον του cluster της σχολής, διαπιστώνουμε ότι η προσπάθειά μας σε αυτή την εργασία στέφθηκε με επιτυχία. Αυτό το κρίνουμε καθώς, όχι μόνο κατορθώσαμε να δημιουργήσουμε μια ολοκληρωμένη παράλληλη εφαρμογή της μεθόδου των γραμμών, που ήταν και το θέμα της εργασίας, αλλά είδαμε στην πράξη τα πλεονεκτήματα της. Αυτά είναι η επιτάχυνση των υπολογισμών και η εξαγωγή των ίδιων αποτελεσμάτων, σε συντομότερο χρονικό διάστημα από την σειριακή εφαρμογή, Mol-C.

4.2 Μελλοντικές επεκτάσεις

Κλείνοντας την παρουσίαση μας, αναφέρουμε μελλοντικές επεκτάσεις που μπορούμε να εκτελέσουμε τροποποιώντας την υλοποίηση που έχουμε αναπτύξει. Οι σκέψεις για διεύρυνση της μέχρι τώρα εφαρμογής ακολουθούν:

- **Αύξηση των διαστάσεων**. Συμφωνά με όσα έχουμε παρουσιάσει στα προηγούμενα κεφάλαια, το πρόβλημα που επιλύουμε αναφέρεται σε δύο διαστάσεις, τη διάσταση του χώρου και αυτή του χρόνου. Ως μια γενίκευση του προβλήματος, μπορούμε να προσθέσουμε περισσότερες χωρικές διαστάσεις. Για παράδειγμα, εισάγοντας μια επιπλέον διάσταση στο χώρο τότε οι μερικές διαφορικές εξισώσεις που επιλύουμε είναι της μορφής:

$$u_t(x, y, z) = f * (u_{xx} + u_{zz}) \qquad (4.2.1)$$

  Με την αύξηση των διαστάσεων του προβλήματος, δεν επιφέραμε αλγοριθμικές μεταβολές στο πρόβλημα. Η διαφορά είναι ότι πλέον δεν αναφερόμαστε σε γραμμές στο επίπεδο, αλλά στο χώρο, με αποτέλεσμα στη διακριτοποίηση των διαστάσεων

του χώρου, να πρέπει να ορίσουμε σημεία που προσδιορίζονται από τρεις συντεταγμένες. Αυτό έχει ως αποτέλεσμα την αύξηση μεγέθους ή των διαστάσεων των πινάκων που συντηρούν την πληροφορία του προβλήματος, και την αύξηση της πολυπλοκότητας, αναφορικά με τις απαιτούμενες για τον προσδιορισμό ενός σημείου πράξεις.

- **Εκτέλεση υπολογισμών χωρίς παύσεις**. Το computation thread, όπως περιγράψαμε νωρίτερα, το διακόπτουμε μετά από κάθε βήμα υπολογισμού, δεσμεύοντας την επεξεργαστική ισχύ για της ανάγκες επικοινωνίας, communication thread. Αντί να διακόπτουμε την εκτέλεση των υπολογισμών, μπορούμε να τη συνεχίσουμε και όταν το thread επικοινωνίας επιστρέψει όλες τις απαραίτητες τιμές που χρειάζεται το computation thread, για να παράγει αποτελέσματα.
- **Εκτέλεση της εφαρμογής σε μεγαλύτερα παράλληλα υπολογιστικά περιβάλλοντα**. Το cluster είναι παράλληλο υπολογιστικό περιβάλλον, το οποίου το πλήθος των παράλληλων διεργασιών περιορίζεται από το πλήθος των επεξεργαστών που το αποτελούν. Ωστόσο, υπάρχουν παράλληλα συστήματα απεριόριστων δυνατοτήτων, όπως είναι το Grid, στα οποία μπορούμε να εφαρμόσουμε την παράλληλη έκδοση της μεθόδου των γραμμών, υψηλών υπολογιστικών απαιτήσεων. Επειδή στα μεγαλύτερα υπολογιστικά περιβάλλοντα οι δυνατότητες κάθε μονάδας επεξεργασίας είναι διαφορετικές, θα πρέπει να κάνουμε αλλαγές στην κατανομή διεργασιών, ανάλογα με τις δυνατότητες του κάθε ενός.

5

Βιβλιογραφία

[1] R. E. SHOWALTER , "*A PDE PRIMER*".

[2] Michael B. Cutlip, *"The Numerical Method of Lines for Partial Differential Equations"*.

[3] "Partial Differential Equation" , http://en.wikipedia.org/wiki/Partial_differential_equation

[4] Vikram Adve, James Browne, Brian Ensink, John Rice, Patricia Teller, Mary Vernon, Stephen Wright,
"An Approach to Optimizing Adaptive Parabolic PED Solvers for the Grid".

[5] John R. Rice and Prathima Rao,
"MOL: METHOND OF LINES APPLICATION".

[6] John R. Rice and Mikel Julan, *"MOL: Method of Lines Application"*

[7] MOL code file site,
http://www.cs.purdue.edu/homes/jrr/MOLfiles/MOL.html

[8] "Method of Lines",
http://www.scholarpedia.org/article/Method_of_lines#Types_of_PDE_Solutions

[9] "The Numerical Method of Lines", http://documents.wolfram.com/mathematica/Built-inFunctions/AdvancedDocumentation/DifferentialEquations/NDSolve/PartialDifferentialEquations/TheNumericalMethodOfLines/Introduction.html

[10] "Σχεδιασμός Παράλληλων Αλγορίθμων" http://inf-server.inf.uth.gr/courses/CE404/

[11] "High performance computing ", http://en.wikiopedia.org/wiki/High_performance_computing,

"Parallel computing", http://en.wikiopedia.org/wiki/Parallel_computing

"Distributed computing", http://en.wikiopedia.org/wiki/Distributed_computing

"Computer cluster", http://en.wikipedia.org/wiki/Computer_cluster

"Grid computing", http://en.wikipedia.org/wiki/Grid_computing

[12] MARZIO SALA, ETH Zurich, W. F. SPOTZ, M. A. HEROUX, "PyThrilinos: High-Performance Distributed-Memory Solvers for Python",

[13] SWIGDocumentation.pdf, http://www.swig.org/Doc1.3/index.html

[14] "Python/C API Reference Manual", http://www.valeriodistefano.com/gnutemberg/python/api.pdf

ΠΑΡΑΡΤΗΜΑ Α. ΕΓΚΑΤΑΣΤΑΣΗ SCIENTIFIC PYTHON MODULE

Εδώ παρουσιάζουμε οδηγό για την εγκατάσταση του Scientific Python στο υπολογιστικό περιβάλλον, όπου σκοπεύουμε να εκτελέσουμε την παράλληλη εφαρμογή που αναπτύξαμε στην διπλωματική εργασία.

Το Scientific module χρησιμοποιεί ως βάση το distutils πακέτο της Python, για την εγκατάσταση και τη μεταγλώττιση του. Το distutils είναι μέρος της πρότυπης βιβλιοθήκης της Python. Αφού κατεβάσουμε το Scientific module, στο φάκελο Doc/PDF του πακέτου βρίσκονται οι οδηγίες εγκατάστασής του, που ακολουθούν:

1. Μπαίνουμε στον εν λόγω φάκελο (στην περίπτωσή μας ScientificPython.2.3) και εκτελούμε τις ακόλουθες εντολές:

   python setup.py build

   python setup.py install

   Η δεύτερη εντολή εγκαθιστά τη ScientificPython στον γονικό φάκελο της Python (/ python2.4/Scientific), που απαιτεί προνόμια διαχειριστή (root privileges). Ωστόσο για να εγκαταστήσουμε την παράλληλη διεπαφή της βιβλιοθήκης MPI, πρέπει πρώτα να είναι περασμένη στο σύστημά μας αυτή η βιβλιοθήκη. Στην περίπτωση του cluster η MPI βιβλιοθήκη είναι περασμένη, οπότε δεν χρειάζεται κάποια επιπλέον διαδικασία.

2. Μεταφερόμαστε στον φάκελο ScientificPython2.3/Include/Scientific/ και αντιγράφουμε τα header files που βρίσκονται μέσα σε αυτόν, με προορισμό την βιβλιοθήκη της Scientific που μόλις δημιουργήσαμε στον χώρο της Python (..//python2.4/Scientific), με τη βοήθεια της εντολής :

   cp *.h ..(μονοπάτι προς το φάκελο της Python μας)/python/Scientific/

3. Πηγαίνουμε στον φάκελο ScientificPython2.3/Src/MPI και πληκτρολογούμε την εντολή :

   python compile.py

   Η εντολή αυτή παράγει ένα εκτελέσιμο mpipython το οποίο πρέπει να αντιγραφεί σε κάθε φάκελο του shell μονοπατιού αναζήτησης. Στην περίπτωσή μας αυτοί οι φάκελου είναι οι /usr/local/bin/ και /usr/bin/ .

   Μετά από αυτή τη διαδικασία το Scientific Python module έχει εγκατασταθεί και είναι έτοιμο να χρησιμοποιηθεί, από την παράλληλη εφαρμογή της μεθόδου των γραμμών, που έχουμε υλοποιήσει.